# Process-based Schema Matching: From Manual Design to Adaptive Process Construction

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet Informatik

vorgelegt von

Diplom-Medieninformatiker
Eric Peukert
geboren am 26. März 1982 in Berlin

Die Annahme der Dissertation haben empfohlen:

1. Prof. Dr. Erhard Rahm (Universität Leipzig)
2. Prof. Dr. Gunter Saake (Universität Magdeburg)

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am 29. November 2013 mit dem Gesamtprädikat *magna cum laude*.

# Acknowledgement

This dissertation was conducted and financed at SAP Research within the research projects Theseus/TEXO and LinkedDesign that were funded by the Bundesministerium für Wirtschaft und Technologie and the European Union. I am grateful that I had the opportunity and time needed to do my research, to publish and finally to write this document. Here I wish to thank Dr. Gregor Hackenbroich for his endless patience and support.

This dissertation could not have been finished without the help and support of my advisor Prof. Dr. Erhard Rahm. I would like to thank him for his strong guidance that he gave me in the numerous discussions we had together. He always pinpointed the strong and weak points of my ideas and encouraged me to improve. When publication deadlines were approaching he always helped, even on week-ends, to review and correct my drafts and gave helpful comments.

I would like to thank my colleagues at SAP and at the database group at the University of Leipzig. In particular, I would like to thank Henrike Berthold for being my advisor at SAP. She helped me to focus on my PhD work and shielded me from daily distractions within our project and recently reviewed parts of this thesis. I also look back onto numerous exiting lunch and office discussions that I had with Konrad Voigt about matching in the fridge. But also within the database group in Leipzig I always felt welcome and part of the team. In particular, I will miss the yearly seminars in Zingst that were always great times. A further thanks goes to the students Julian Eberius and Veronika Thost for their work and the fruitful discussions we had. I highly appreciated to work with them.

Finally, I would like to thank my family and friends that endured my strong absence in the last years. I hope that some still know me. And most importantly I would like to thank my wife Judith for her love and support. This already began at the TU Dresden years before the dissertation when preparing for our first exams. This made times of work much more enjoyable and rewarding.

Leipzig, den 6. Juni 2013 Eric Peukert

# Abstract

Mappings between complex metadata structures like database schemas, XML schemas or ontologies are needed in a number of domains such as data integration, ontology alignment or web service composition. A mapping describes how elements of one metadata structure correspond to elements of another metadata structure. Defining such mappings is a complex and time-consuming process. It is often done manually, with the help of point and click interfaces.

In the last 10 to 15 years a strong effort was made in research to partly automate the mapping process. Many schema- and ontology matching systems were developed to semi-automatically compute mapping suggestions for a user. Most systems are build and tuned for a specific domain of mapping problems. They are often not robust enough to be able to cope with different mapping problems without high configuration effort. Moreover, existing sytems often face performance issues when mapping large structures. This thesis investigates how to support the user in the task of configuring schema matching systems and how to improve run-time performance.

Initially, fundamental concepts of schema matching are introduced and an overview to the existing body of work is given. In a pre-evaluation of existing approaches further research on the configuration aspect is motivated. In particular, approaches that support manual tuning and that (partially) automate the configuration and construction of schema matching systems are reviewed.

The second part of the thesis then introduces a new matching process model that supports adaptivity. It defines a set of operators for matching and filtering that can be used to create domain-specific matching processes. A condition construct allows a user to introduce adaptive behavior in a matching process. Such conditions help to adapt the match processing to the specifics of the problem at hand based on features of the input schemas or intermediate matching results.

A new tool for graphically constructing and tuning matching processes is introduced. It eases development of matching processes by using a drag and drop metaphor. Furthermore, it provides visualizations for tuning and debugging matching processes. From the experience that was made by modeling matching processes a number of reappearing matching process design patterns are identified. They could serve as a basic library of templates for constructing new matching processes.

In a third part, the thesis introduces a novel rule-based technique for automating the construction and configuration of matching processes. The configuration of

run-time performance aspects is automated by rewriting matching processes. A number of filter-based rewrite rules are presented. Based on a simple cost-model, parallel combinations of matchers can be rewritten to sequential matching processes containing filter operators. By sequentializing parallel matching processes with filter-based rewrite rules significant run-time performance improvements (up to a factor of 9) could be achieved. In particular together with a so-called dynamic filter strategy improvements were achieved without changing the quality of a schema matching process.

The rewrite-based approach is adopted for automatically constructing matching processes that are tailored to a given mapping problem. Based on measured features of the input schemas and intermediate results so-called matching rules can be defined. These rewrite rules rely on analyzing the input schemas and intermediate results while executing a process and rewrite the process to better fit to the problem at hand. The evaluation shows that the approach behaves more robust than existing schema matching approaches without involving the user in complex configuration tasks. The final system is self-configuring so that is does not need other input than the input schemas to compute a mapping.

# Contents

**I Introduction** **1**

**1 Introduction** **3**

1.1 Motivation . . . 3
1.2 Scientific Contribution . . . 6
1.3 Structure of the Thesis . . . 7

**2 Schema Matching Basics** **9**

2.1 Introduction . . . 9
2.2 Matchers . . . 13
2.2.1 String-based Techniques . . . 14
2.2.2 Constraint-based Techniques . . . 15
2.2.3 Mapping and Structure Reuse . . . 15
2.2.4 Instance-based Techniques . . . 16
2.2.5 Graph-based Techniques . . . 16
2.2.6 Logic-based Techniques . . . 17
2.3 Combination of Matcher Results . . . 18
2.3.1 Triangular Norms & Minkovski Distances . . . 19
2.3.2 Weighted Combination . . . 19
2.3.3 Majority Voting . . . 20
2.4 Selection of Correspondences . . . 21
2.4.1 Threshold-based Strategies . . . 21
2.4.2 Maximum-based Strategies . . . 22
2.4.3 Optimization-based Strategies . . . 23
2.4.4 Selection with Background Knowledge . . . 23
2.4.5 Combining Selection Strategies . . . 24
2.5 Comparing Combination and Selection Strategies . . . 24
2.5.1 Conclusions . . . 28
2.6 Strategies used in this Thesis . . . 28

**3 Configuration of Matching Systems and Adaptivity** **31**

3.1 Matching System Topologies . . . 31
3.1.1 Parallel . . . 32
3.1.2 Sequential . . . 32

3.1.3 Iterative . . . 33
3.1.4 Hybrid Approaches and Workflows . . . 33
3.2 Automating the Configuration . . . 34
3.2.1 Machine Learning Approaches . . . 35
3.2.2 Feature- and Rule-based Approaches . . . 36
3.3 UI Support for Matching and Configuration . . . 38
3.3.1 Schema- and Mapping Visualization . . . 39
3.3.2 Interactive Schema Matching . . . 41
3.3.3 Visualization Support at Design-Time . . . 43
3.4 Improving Performance in Schema Matching . . . 45
3.4.1 Divide and Conquer . . . 45
3.4.2 Filtering Schema Parts . . . 46
3.4.3 Avoiding Repetitions . . . 46
3.4.4 Improved Data Structures . . . 46
3.4.5 Process-based Performance Optimization . . . 46
3.5 Comparative Summary of Matching Systems . . . 47

**II Process-based Schema Matching 51**

**4 Adaptive Matching Process Model 53**
4.1 Matching Process Definition . . . 53
4.2 Operators of the Matching Process . . . 54
4.2.1 Import/Export Operators . . . 54
4.2.2 Match Processing Operators . . . 56
4.2.3 Control Structures . . . 59
4.2.4 Further Operators . . . 62
4.3 Features and the Condition Operator . . . 63
4.3.1 Schema Features . . . 65
4.3.2 Schema Similarity Features . . . 68
4.3.3 Matrix Features . . . 69
4.3.4 Matrix Similarity Features . . . 74
4.3.5 Example . . . 75
4.4 Matching Process Design Patterns . . . 76
4.4.1 Identified Set of Patterns . . . 76
4.4.2 Evaluating Patterns . . . 79
4.5 Matching Process Execution Framework . . . 80
4.5.1 Matching Process Execution . . . 81
4.6 Advances over Related Work . . . 83

**5 Graphical Modeling of Schema Matching Processes 85**
5.1 Graphical Modeling of Processes . . . 85
5.2 Tuning of Matching Processes and Analysis of Intermediate Results . 87
5.3 Comparative Evaluation of Matching Processes . . . 88

5.4 Advances over Related Work . . . . . . . . . . . . . . . . . . . . . . . . . 91

**III Rewrite-based Process Tuning and Construction 93**

**6 Performance Oriented Matching Process Rewrite 95**
6.1 Comparison Filtering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95
6.1.1 Static Threshold-based Strategy . . . . . . . . . . . . . . . . . 96
6.1.2 Relative Threshold-based Strategy . . . . . . . . . . . . . . . . 96
6.1.3 Static TopN Strategy . . . . . . . . . . . . . . . . . . . . . . . . 97
6.1.4 Dynamic Threshold-based Strategy . . . . . . . . . . . . . . . . 97
6.1.5 Dynamic Delta-based Strategy . . . . . . . . . . . . . . . . . . 99
6.2 Evaluating Sequential Matcher Combinations . . . . . . . . . . . . . 100
6.2.1 Incidence Graph . . . . . . . . . . . . . . . . . . . . . . . . . . 101
6.3 Matching Process Rewrite Technique . . . . . . . . . . . . . . . . . . 102
6.3.1 Filter-based rewrite rules . . . . . . . . . . . . . . . . . . . . . 104
6.3.2 Applying Filter-based Rules . . . . . . . . . . . . . . . . . . . . 106
6.3.3 Further Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107
6.3.4 Matching Process Rewrite System . . . . . . . . . . . . . . . . 108
6.4 Advances over Related Work . . . . . . . . . . . . . . . . . . . . . . . . 109

**7 Adaptive Schema Matching based on Rules 111**
7.1 Rule Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111
7.2 Adaptive Process Construction . . . . . . . . . . . . . . . . . . . . . . . 112
7.2.1 Staged Execution . . . . . . . . . . . . . . . . . . . . . . . . . . 113
7.2.2 Executing a Stage . . . . . . . . . . . . . . . . . . . . . . . . . . 116
7.2.3 Termination and Iteration . . . . . . . . . . . . . . . . . . . . . 116
7.2.4 Stepwise Matching Process Execution . . . . . . . . . . . . . . 117
7.3 Rule Collection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
7.3.1 Refine Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
7.3.2 Refine Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
7.3.3 Rewrite Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
7.3.4 Combination Rules . . . . . . . . . . . . . . . . . . . . . . . . . 126
7.3.5 Selection Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . 127
7.4 Adaptive Execution Examples . . . . . . . . . . . . . . . . . . . . . . . 129
7.4.1 Simple Order Example . . . . . . . . . . . . . . . . . . . . . . . 129
7.4.2 Simple Order Example - Changed Structure . . . . . . . . . . 131
7.4.3 Example 3 - Mouse-Anatomy to NCI Thesaurus . . . . . . . . 132
7.5 Adaptive Matching System . . . . . . . . . . . . . . . . . . . . . . . . . 134
7.6 Discussing Design Decisions . . . . . . . . . . . . . . . . . . . . . . . . 135
7.7 Improvement over Related Work . . . . . . . . . . . . . . . . . . . . . 137

**IV Evaluation 139**

**8 Evaluation 141**
8.1 Data Set . . . 142
8.2 Evaluating Rewrite-based Performance Optimization . . . 144
8.2.1 Performance Comparison Results . . . 145
8.2.2 Influence of Threshold on Execution Time . . . 147
8.3 Evaluation of the Adaptive Rewrite-based Schema Matching Approach 148
8.3.1 Robustness of Overall Matching Approach . . . 150
8.3.2 Influence of Individual Rules . . . 153
8.3.3 Process Heterogeneity . . . 156
8.3.4 Monogamy-based Termination of Iterations . . . 156
8.4 Summary of Evaluation . . . 158

**V Summary and Outlook 159**

**9 Summary and Outlook 161**
9.1 Summary . . . 161
9.1.1 Process-based Schema Matching . . . 161
9.1.2 Automatic Configuration and Construction of Matching Processes162
9.2 Outlook . . . 162

**A Feature Collection 165**
A.1 Schema Features . . . 165
A.1.1 String-Meaningfulness Feature . . . 165
A.1.2 Element-Token-Ratio Feature . . . 166
A.1.3 Repeating-Elements Feature . . . 166
A.1.4 Repeating-Fragments Feature . . . 166
A.1.5 Schema-Depth Feature . . . 167
A.1.6 Path-Variance Feature . . . 167
A.2 Schema Similarity Features . . . 167
A.2.1 Feature-Similarity and Average . . . 167
A.2.2 Similar-Language Feature . . . 168
A.2.3 Structural-Similarity Feature . . . 168
A.3 Matrix Features . . . 169
A.3.1 Selectivity Feature . . . 169
A.3.2 Cross-Matches Feature . . . 169
A.3.3 Node-Position Feature . . . 170
A.3.4 Multi-Matches Feature . . . 170
A.3.5 Sibling-Distribution Feature . . . 171
A.4 Matrix Similarity Features . . . 171
A.4.1 Commonality Feature . . . 171
A.4.2 Complementarity Feature . . . 172

**B Rule Collection** **173**
B.1 Starting Refine Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . 173
B.1.1 Add-Statistics Rule . . . . . . . . . . . . . . . . . . . . . . . . 173
B.1.2 Add-Annotation Rule . . . . . . . . . . . . . . . . . . . . . . . 174
B.1.3 Add-Instance Rule . . . . . . . . . . . . . . . . . . . . . . . . . 174
B.1.4 Add-Restriction Rule . . . . . . . . . . . . . . . . . . . . . . . 175
B.2 Refine Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 175
B.2.1 Add-Leaf Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . 175
B.2.2 Add-Sibling Rule . . . . . . . . . . . . . . . . . . . . . . . . . . 176
B.3 Rewrite Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 177
B.3.1 Noise-Filter Rule . . . . . . . . . . . . . . . . . . . . . . . . . . 177
B.3.2 Blocking-Filter Rule . . . . . . . . . . . . . . . . . . . . . . . . 177
B.4 Selection Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178
B.4.1 Max1-Select Rule . . . . . . . . . . . . . . . . . . . . . . . . . . 178
B.4.2 Adaptive-Threshold Rule . . . . . . . . . . . . . . . . . . . . . 179
B.4.3 Restrict-to-N:N . . . . . . . . . . . . . . . . . . . . . . . . . . . 180

**C Evaluation Data** **181**
C.1 Data Set . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 181
C.2 Incidence Graph from Evaluation . . . . . . . . . . . . . . . . . . . . . 183
C.3 Rewritten Matching Processes . . . . . . . . . . . . . . . . . . . . . . . 184

**Bibliography** **187**

# Part I

# Introduction

# Chapter 1

# Introduction

## 1.1 Motivation

We are faced with an increasing number of heterogeneous corporate datasources and datasources on the Web. Some people say that "data is the new oil" [1] which refers to the increasing value of data for businesses to take decisions. But, similar to crude oil that needs to be refined, data from different sources needs to be brought into context, merged and transformed to allow for integrated analysis which finally creates value. However, this process is complex and costly since datasources rely on heterogeneous data formats within independently developed IT-systems. To exchange data between different IT-systems or for integrating data, mappings between the metadata structures that describe each datasource are needed. A mapping describes how elements of one metadata structure correspond to elements of another metadata structure. Within this thesis a mapping is soley a set of such correspondences.
Commonly used metadata structures are relational database schemas, XML schemas or ontologies. Relational database schemas describe the attributes and relations of entities in a database. XML Schemas are used to describe hierarchically structured XML documents which can be stored or exchanged as messages. Ontologies are similar to schemas but model data on a higher level of abstraction.
In the following, several classical application areas are described where mappings between such metadata structures are needed:

- When companies merge, their IT-systems and underlying databases often need to be integrated. In many cases databases are integrated by building a federated system with a global schema [153, 96, 15]. Existing database schemas need to be mapped to this global schema and queries posed to the federated system are translated to the local databases. Due to the heterogeneity of the involved schemas, creating the mappings is often complex.

- Data migration is often required when data is moved from a legacy system to a new target system such as migrating from SAP R3 to SAP Business By Design.

[1] Citation by Clive Humby, 2006

In that context a mapping between the legacy database and the new database is required. With the help of the mapping, transformation code can be written. This transformation code is executed once until all data is transferred correctly.

- Data is often explored and analyzed within central data warehouses. It is loaded into the warehouse with the help of programmatic transformation scripts such as ETL (Extract Transform Load). Again, for constructing transformation scripts, mappings from source schemas to the warehouse schema are crucial.

Apart from database integration and migration a number of novel application areas recently evolved that also strongly rely on predefined mappings like catalog integration, service oriented architectures and the semantic web:

- Catalog integration is faced with the problem of integrating product catalogs from multiple market places such as Amazon or Ebay. Product catalogs rely on hierarchies of categories to group offered products. In order to get an overview to multiple catalogs, product catalogs need to be mapped and integrated [5, 81].

- In novel service oriented architectures, business processes cross the organizational boundaries and integrate services from external service providers. Instances of such services can be payment, travel accounting or analytics. This involves exchange of data and messages with external IT-systems. To implement the communication and exchange, outputs of internal services need to be mapped to inputs of external ones. Even with standardized data schemas such as the Unified Business Language (UBL) [27], the Core Components Specification (CCTS)[1, 83] or ebXML [92] the message mapping remains a complex step when building business processes.

- According to the vision of the Semantic Web [21] information on the Web should be represented and described in a form that simplifies processing and integration. For that purpose, ontologies are commonly used. Ontologies help to describe rules, processes and terminology in a human and machine readable way. Examples for ontologies are the Semantic Web Service Ontology [172] or the life science ontologies such as the Foundational Model of Anatomy (FMA) [147] and the Gene Ontology (GO) [10]. The latter is used to describe complex biological processes. In order to integrate information from sources that where described by independently modeled ontologies, mappings between ontologies need to be defined. Due to the large size of such ontologies (>34000 terms), this task is challenging.

All these different application areas have in common that they crucially rely on mappings. Defining such mappings is in most cases a complex, time-consuming process. It is often done manually with the help of point and click interfaces. Even if two metadata descriptions, for example two database schemas intend to describe similar data, they often differ in the structure and naming of tables, attributes and used types. Due to the involved heterogeneity expert knowledge is needed.

In the last 10 to 15 years a strong effort was made in research to partly automate the mapping process. It turned out, that finding an executable mapping can be splitted in three steps of first identifying corresponding elements (matching) then defining an executable transformation specification and finally testing and executing those transformation scripts (see Figure 1.1).

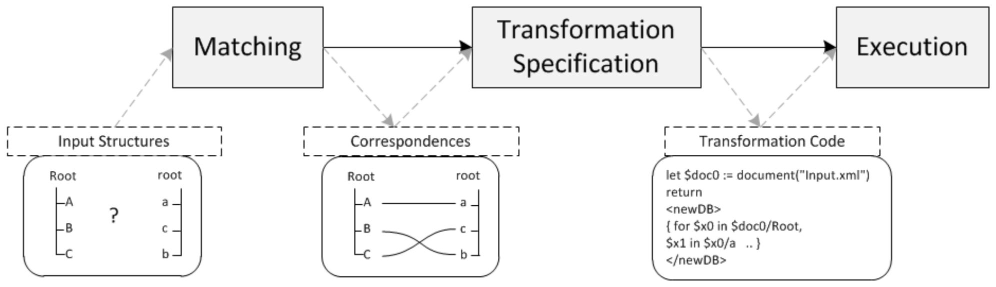


Figure 1.1: Matching/Mapping Process

These steps can also be labeled mapping discovery, mapping representation and execution [178]. Some mapping tools support the whole process like the well-known research tool Clio [59]. It performs matching, helps to create transformation code and executes transformations of instances by using XQuery. In some application areas such as Ontology Alignment only the first step of finding correspondences is relevant since no data needs to be transformed.

This thesis primarily focuses on the matching phase. In particular, improving existing approaches that semi-automatically compute correspondences is of major interest. There are already many of such matching systems often designed for specific domains and labeled accordingly as data base schema matching systems, XML schema matching systems, ontology matching systems or model matching systems. Surprisingly, every year a number of additional matching systems are proposed in the OAEI Contests [54]. But, until now matching systems are only rarely applied in practice with few exceptions of Microsoft BizTalk Mapper [24], IBM Clio [59] or Altova Mapforce [9]. There are multiple reasons for that:

1. Matching systems are often build and tuned for a specific domain of mapping problems. They are not robust enough to be able to cope with different mapping problems without high configuration effort. Configuring and tuning an existing schema matching system still seems more complex than rebuilding it from scratch. In particular, user support in the configuration task is urgently needed. According to Gal and Shvaiko self-tuning of matching systems is “largely unexplored” [63].

2. Existing systems are often monolithic and are not easy to extend. For instance, COMA++ [11] and AgreementMaker [31] only provide binary versions of their

matching tools (recently the COMA++ sources where made publicly available). Extending them by additional components is cumbersome if not impossible.

3. Reusing and comparing parts of existing matching systems is complicated since matching techniques are often bundled with system internal data structures. Moreover, for many systems the sources are not available. But even if sources are available, the wish for ownership and control of the code to simplify parameter tuning seems to be strong.

4. New matching applications have additional run-time performance requirements that are often not met by existing systems. In many cases an automatically computed mapping only serves as suggestion which can be refined by a user. Thus, matching systems need to compute mappings fast in order to not disrupt the workflow of the user, even for large size mapping problems. However, improving the performance of an existing matching system often breaks the monolithic architecture and involves high programming effort.

All these points already highlight major challenges in schema matching. Some of them were also discussed in a recent book on schema matching [18] and a paper from Shvaiko and Euzenat [132].

## 1.2 Scientific Contribution

In this thesis, a number of contributions are made to alleviate some of the aforementioned problems:

1. A matching process model is introduced that allows designing domain specifc matching processes with a set of standard operators. Through condition and filter operators, the robustness of a matching process can be increased. Additionally, some operators help to improve run-time performance.

2. A matching framework is presented that can easily be extended by new or existing matching techniques. The framework simplifies comparisons of existing components in a common environment to help a user to select most appropriate matching approaches for a given mapping problem.

3. A graphical user interface is used to ease development of matching processes by using a drag and drop metaphor. It provides tools and visualizations for tuning and debugging matching processes.

4. From the experience that was made by modeling matching processes a number of reappearing matching process design patterns are identified. They could serve as a basic library of templates for constructing new matching processes.

5. To further automate the configuration of run-time performance aspects, a new approach is introduced that automatically rewrites matching processes to

increase performance. A number of filter-based rewrite rules are presented that are able to significantly improve the performance of a given matching process, some without influencing the quality.

6. Finally, the rewrite-based approach is adopted for automatically constructing matching processes that are tailored to a given mapping problem. Based on measured features of the input schemas and intermediate results a matching process is adaptively constructed and configured. The evaluation shows that the approach behaves more robust than existing schema matching approaches without involving the user in complex configuration tasks.

## 1.3 Structure of the Thesis

In **Chapter 2** an overview to schema matching basics is given and further research for supporting the configuration of matching systems is motivated. **Chapter 3** reviews approaches to support or to automate the configuration of schema matching processes or to optimize performance in schema matching. The core of the thesis is then structured in three parts. The first part - Process-based Schema Matching - introduces a process model that allows a user to manually construct and execute matching processes. It consists of the following chapters:

**Chapter 4** introduces a new process model with a number of operators that can be used to design adaptive matching processes. The chapter describes how adaptive matching processes can be built by using conditions and features. Features of the input schemas and intermediate results of a matching process are presented. These features are later used to implement a self-configuring matching system (see Chapter 7). A novel matching framework is described that allows us to plug-in and compare existing matching components in a common environment.

**Chapter 5** introduces a tool for graphical modeling of matching processes. The tool allows a user to investigate intermediate results of matching processes which helps to interactively tune a matching process. Based on the modeling experience, commonly used patterns for developing matching processes are described and evaluated.

In the second part - Rewrite-based Process Tuning and Construction - an approach is presented for automated tuning and construction of matching processes by using rewrite rules. It consists of the following two chapters:

**Chapter 6** introduces a rewrite-based approach to increase the performance of given matching processes. The approach applies rules and filter operators to significantly reduce element comparisons while matching.

**Chapter 7** adopts the idea of rewrite rules from Chapter 6 and presents an automatic rule-based approach to create a matching process based on computed features of the input schemas and intermediate matching results.

The last part - Evaluation - evaluates the approaches. The rewrite-based performance optimization and the adaptive matching process construction approach are evaluated in **Chapter 8** on a heterogeneous set of mapping problems from different domains. The goal is to show that the rewrite-based performance optimization approach is able to significantly improve performance. The rule-based construction of matching processes should result in a robust matching system that achieves good quality in most test cases that are used for evaluation. Afterwards, the results are summarized and future work is discussed in **Chapter 9**.

# Chapter 2

# Schema Matching Basics

## 2.1 Introduction

There is lively research in the context of matching metadata structures. With each new application area additional matching requirements are identified and existing approaches are adapted. Unfortunately, similar or equal concepts are often named differently and sometimes terminologies conflict between research groups of a similar domain. Even the mapping task itself has multiple names depending on the type of metadata structure to be matched. It can be labeled XML- or relational schema matching, ontology alignment [57], (meta-) model matching [61], link discovery [124] or process matching [184]. And still, further names can be found.

A review of related work on database schema, XML, meta-model and ontology matching reveals that for all these domains similar matching techniques are actually used with exceptions of some domain specific matching approaches like reasoning-based techniques in ontology matching [115, 116, 179] or meta model specific graph matchers [170]. Shvaiko and Euzenat made similar observations in their book on ontology matching that "Schemas and ontologies share similarity since they both provide a vocabulary of terms and somewhat constrain the meaning of terms used in the vocabulary. Hence, they often share similar matching solutions" [132]. To simplify the review of related work, the term schema is relaxed in this thesis to refer to any metadata structure that can be matched. It could also be named "model" as proposed by Bernstein and Melnik [22] but since schema matching is the older research field, the term "schema" is used. One can easily argue about relaxing the schema term given the obvious differences of the described metadata representations. However, abstracting from individual representations into a common generic representation can help to foster reuse of matching techniques across domains as is later shown in the contributions of this thesis. In the following, the key terminology and building blocks of schema matching systems are introduced.

Most metadata structures can be abstracted to a common representation that we call a *Schema*:

**Definition 1.** ***(Schema)*** *A Schema $S$ consists of $|S|$ Schema Elements $s_1, \ldots, s_{|S|}$. Each schema element is described by a number of attributes such as name, type, annotation, instances or cardinality. With $s.n$, $s.t$, $s.a$, $s.i$ and $s.c$ the name, type, annotation, instances and cardinality of an element $s$ is referred to. Schema elements are connected with each other and build a directed acyclic graph structure. Schema elements can have a number of children and possibly multiple parent elements. Moreover, instances can be attached to schema elements which adhere to the structure of the schema element graph.*

This definition of a schema is generic and allows representing the most important information of existing structures that is needed for matching.

**Definition 2.** ***(Schema Matching)*** *Schema Matching is the process of discovering semantic correspondences between schema elements. It takes at least a source schema S and a target schema T as input and results in a Mapping M.*

The task of matching more than two schemas is often called holistic schema matching [74, 159]. This thesis focusses on matching only two schemas at a time.

**Definition 3.** ***(Mapping)*** *A Mapping M between two schemas $S$ and $T$ consists of a set of correspondences. Correspondences are triples $(s, t, v)$ with $s \in S$ and $t \in T$ and $v$ representing the similarity between s and t as a value in the interval $[0, 1]$. With $sim\,(s, t)$ the similarity value $v$ of a schema element pair $(s, t)$ is referenced.*

The defintion of correspondences within this thesis only allows having one source and one target element per correspondence and a correspondence type is left out. Complexer mapping models could be used to represent mapping hierarchies, n:m correspondences and transformation functions. Such complex mapping models are typically valuable when post-processing computed mappings as often done in ontology matching systems like ASMOV [86] or S-MATCH[66]. In this thesis, the primary focus was set onto the match processing so that pre- and postprocessing and therefore a complex mapping model was not needed. Moreover, most existing matching techniques are not able to identify complex correspondences that map multiple source elements to a single target element or vice versa.
Nevertheless, elements of the source or target schema could be part in no, one or many correspondences which creates mappings of different cardinalities. Within this thesis, the UML notation to refer to cardinalities [118, 117] is used. A mapping in which every source element and every target element can only find no or one partner is represented as [0,1][0,1] mapping meaning that elements in the source take part in 0 or 1 correspondence and elements of the target also take part in 0 or 1 correspondence. Such a mapping can also be called a one-to-one mapping. Mappings that contain multiple partners for source or target elements are called multi-mappings. There can be different kinds of multi-mappings such as one-to-many [0,1][0,*], many to one [0,*][0,1] or many-to-many [0,*][0,*] mappings.

In literature also other terms are used to refer to a mapping such as Matchings [62], Alignment [52, 51, 126] or Matching Result [170]. Some also refer

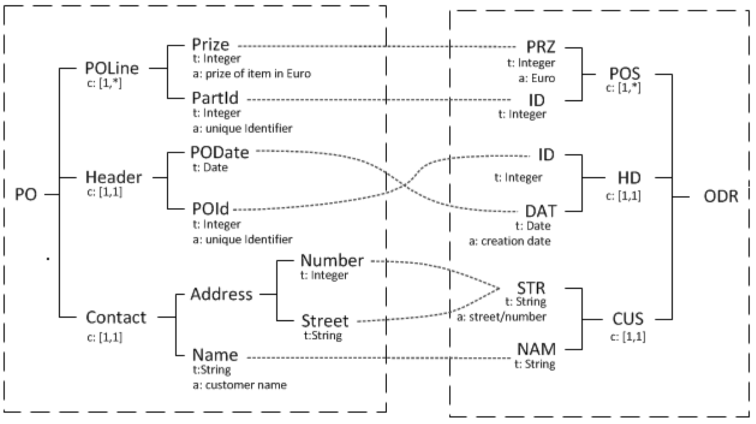


Figure 2.1: Schema and mapping example

to sets of Matches instead of correspondences. Within this thesis, a distinction between a mapping and an executable transformation specification is made. This is important since a number of groups refer to transformation specifications also with the term "Mapping" [59, 112]. These specifications consist of logical expressions that can be translated to executable queries or transformation scripts (see Clio for an example [59]).

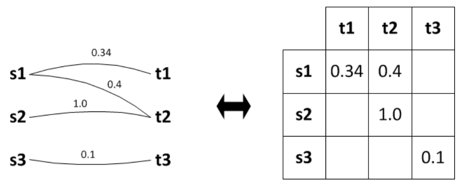
In Figure 2.1 two exemplary schemas are shown that describe purchase orders. The left schema contains elements with partly abbreviated names. Some elements contain additional information such as cardinality, type or annotation. The schema on the right is similarly structured. The names of elements are abbreviated to three letters as was often done within legacy database schemas. The dotted lines between the individual schema elements are corresponding elements. To improve readability, correspondences between parent elements are not visualized.

A large body of work in schema matching tries to automate the matching process. This is done by schema matching systems, which could also be named differently depending on the type of metadata structure to be matched as discussed above. The definition within this thesis is as follows:

**Definition 4.** ***(Schema Matching System)*** *A Schema Matching System automates the schema matching task. It relies on a set of matching operations (matcher, combination and selection) to compute a mapping.*

**Definition 5.** ***(Matcher)*** *The Matcher operation gets two input schemas and optionally a mapping as input and computes a mapping as output. It maps pairs of elements of the input schemas on values between 0 and 1: $s \times t \rightarrow [0, 1]$ with $s \in S$ and $t \in T$. This value represents the confidence assigned by the matcher operation to the pair $(s, t)$, which is called similarity value.*

In literature, matchers are sometimes also called matching technique [57], similarity measure [45, 88], matching algorithm, learner, matching approach, voter [152],



| | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 0.34 | 0.4 | |
| s2 | | 1.0 | |
| s3 | | | 0.1 |

Figure 2.2: Example mapping and similarity matrix

or even distance metric [124]. Within this thesis, these terms are sometimes used interchangeably. Implementations of matchers mostly rely on heuristics which will be explained in detail in Section 2.2. Many matching systems internally represent a mapping as a so-called similarity matrix:

**Definition 6.** ***(Similarity Matrix)*** *A similarity matrix $SM$ is a matrix that consists of $|S| \times |T|$ cells. Each entry $sm_{i,j}$ references a pair of the i-th element of the source schema and the j-th element of the target schema and represents the degree of similarity between them. Each similarity matrix $SM$ consists of $|S|$ rows $\left\{r_1, \ldots, r_{|S|}\right\}$ and $|T|$ columns $\left\{c_1, \ldots, c_{|T|}\right\}$.*

An exemplary mapping and corresponding similarity matrix is shown in Figure 2.2

Typically multiple matchers are used and their results are combined. For that purpose a *Combination* operation can be used.

**Definition 7.** ***(Combination)*** *The Combination operation takes multiple mappings $m_1, \ldots, m_K$ as input and computes a mapping as output. It relies on some combination strategy that maps multiple similarity values for a given pair of elements to a single similarity value.*

The combination result still contains similarity values for each pair of schema elements from the source and target schema. In order to extract the most probable correspondences a *Selection* operation is applied.

**Definition 8.** ***(Selection)*** *The Selection takes a mapping as input and extracts the most probable correspondences for an output mapping. It relies on a selection strategy to compute a new mapping $M'$.*

For evaluating the behaviour of a schema matching system and the quality of computed results the f-measure value is typcially used. Given a correct mapping which could be called gold standard, reference mapping or reference alignment the f-measure can be computed. It relies on the harmonic mean of precision and recall:

$$f\text{-}measure = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.1}$$

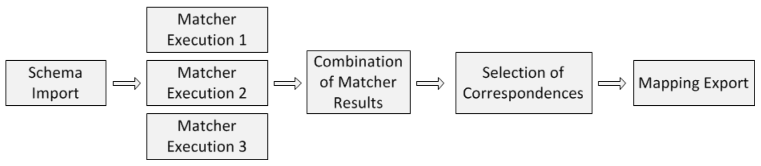


Figure 2.3: Common internal workflow of a schema matching system

Precision is defined as the ratio of found correct correspondences to the number of found correspondences in a result mapping. Recall computes the ratio of found correct correspondences to the number of all correct correspondences.

Figure 2.3 summarizes the workflow of a generic matching system as it was proposed in COMA [35]. Initially, source and target schemas are imported and transformed into some internal representation. Then a number of matchers are executed that each compute similarities for each pair of source and target schema elements. Results from individual matchers are combined and the most probable correspondences are selected. Finally, the computed mapping is transformed to some external format. The workflow can be very different in existing matching systems but the operations (matcher, combination and selection) are commonly used. In the following sections, existing matchers as well as combination and selection strategies are reviewed and key approaches are explained.

## 2.2 Matchers

Many matching techniques were proposed in the literature and still additional approaches are added. In order to get an overview multiple classifications of existing techniques were created [57, 155, 141]. Rahm and Bernstein first introduced a classification separating individual matching techniques from compositions of matching techniques [141]. The classification of individual matching techniques was later extended by Euzenat and Shvaiko to include ontology-specific approaches [155, 57]. There are many criteria that can be used for classification. Most of the existing ones rely on two orthogonal criteria that are the kind of input and the granularity of the input:

- *Kind of Input* describes what input a matcher uses for computing similarities. Among others, Rahm and Bernstein [141] distinguished between schema-based and instance-based information.
- *Granularity* describes the difference between element-level and structure-level matchers. Element-level matchers only compute similarities based on individual elements whereas structure-level matchers incorporate neighboring elements into the similarity computation.

A simplified version of the classification from Rahm and Bernstein is shown in Figure 2.4. It also lists the most important classes of basic matching techniques with some

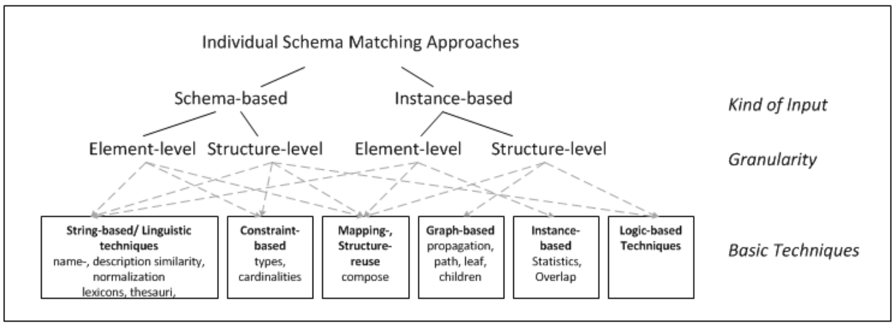

Individual Schema Matching Approaches
Schema-based
Instance-based
Kind of Input
Element-level
Structure-level
Element-level
Structure-level
Granularity
String-based/ Linguistic techniques name-, description similarity, normalization lexicons, thesauri,
Constraint-based types, cardinalities
Mapping-, Structure-reuse compose
Graph-based propagation, path, leaf, children
Instance-based Statistics, Overlap
Logic-based Techniques
Basic Techniques

Figure 2.4: Simplified classification of matching techniques from [141] with adaptations

sample approaches as was proposed by [57]. In the following, each of these classes is described in more detail.

### 2.2.1 String-based Techniques

Most string-based techniques interpret the input syntactically. Element names, type-names or annotations are solely treated as sequences of characters. For computing the similarity of character sequences a number of so-called string similarity functions or string distance functions where proposed in the area of record linkage [93], similarity search [180], information retrieval [144] and schema matching. The simplest string similarity measure performs exact string matching. However, in schema matching approximate string matching techniques are more relevant. They can be classified as edit-distance like, token-based and hybrid approaches [29].

- Edit-based techniques compute the distance between strings based on the number of edit operations that are needed to transform a string $A$ into a string $B$. Popular measures are the Levenshtein (often referred to as Edit-Distance) [73], Jaro [84] and Jaro-Winkler metric [174]. The latter rely on the number and order of common characters as well as the longest common prefix. Further representatives are Monge-Elkan [122], and Smith-Waterman [158].

- Token-based techniques split a string into tokens or words and compute a similarity based on the set of tokens/words. The set comparison is often done using the Jaccard-similarity which is computed with $\frac{S \cap T}{S \cup T}$ for two sets of tokens $S$ and $T$. The set of tokens can be built as N-grams which are all character sequences of length N within a string [103]. For annotations, which are typically longer strings the TFIDF [146] measure from information retrieval is often used. TFIDF counts exactly matching words and gives each match a weight. The weights are computed from word frequencies within a text compared to frequencies within a whole text corpus. In other words, TFIDF identifies the most characteristic words in an annotation.

- Hybrid techniques combine multiple string similarity techniques. For example TFIDF can be extended to include not exactly matching word pairs which is called Soft-TFIDF [29]. Also the Name matcher from COMA [35] is a hybrid technique since it splits strings into word tokens, computes pairwise word-token similarities and combines these word-similarities back to a single similarity value. In the preparatory phase of this thesis the Name matcher was extended by a TFIDF-like word token weighting [138] which solves problems of repeating word tokens of multi-word schema element names within a single schema. Similar token weighting was also proposed for object matching [165].

Some techniques (often called linguistic techniques) rely on a language model that treat strings as natural language objects that can be interpreted and translated. There are intrinsic methods that mainly cope with normalization of strings like tokenization, term extraction, stop-word removal or stemming. Extrinsic methods rely on thesauri, lexicons and dictionaries such as Wordnet [119] that can be used for translation or expansion of abbreviations. Language-based techniques are often performed before applying string-based techniques to correct, normalize and expand element names and annotations.

### 2.2.2 Constraint-based Techniques

When defining schema elements, additional constraints can be added such as types, allowed values, restrictions of value ranges, optionality or cardinalities. Euzenat and Shvaiko [57] refer to internal structure when talking about such constraints. Often, a comparison table approach is used with fixed similarities for simple types [35, 57, 96]. More advanced techniques directly compare cardinalities, restriction entries and value ranges by intersection and containment computation. Constraint-based techniques are particularly useful for restricting the search space since they do not serve well as a standalone matching technique [36]. In some matching systems constraint matching is used as a post processing step such as removing wrong type matches [66].

### 2.2.3 Mapping and Structure Reuse

A promising set of new techniques relies on mapping and structure reuse. A recent overview from Rahm to existing techniques can be found in [140]. Existing mappings are typically stored in a repository for later reuse. They can either be retrieved directly or new mappings can be derived. For deriving mappings, transitivity of mapping paths [35] are often exploited. But also learning-based [39, 38], probabilistic [125] and reasoning-based techniques were proposed.
**Schema-based approaches** save complete mappings along with their source and target schemas which builds a repository of mappings [182, 130]. To retrieve mappings the source and target schemas are entered as input and the stored mapping is returned. The approach can be extended by exploiting transitivity of mapping paths as described in [35, 4].

**Element-based approaches** store schemas and mappings on element level. The structure of schemas is mostly ignored. Simple examples of that kind are synonym dictionaries that are used in a number of systems [35, 39, 66, 152, 150]. Again, transitivity can be exploited or machine learning derivation techniques are used [102] that use element mappings as training data to tune a set of matchers.
**Global schema approaches** store mappings from source schemas to a global mediated schema. To solve a new matching problem between two schemas the transitive relation passing through the global schema is exploited. A global schema can be fixed or constructed automatically based on some merging algorithm [40, 143, 150]. The process of mapping schemas to a global schema can also be called anchoring, contextualization or lifting when dealing with upper-level ontologies [57]. If multiple ontologies are used as background knowledge composition paths can be constructed [70].
**Fragment-based approaches** decompose schemas and mappings into fragments [149]. This is beneficial in large scale schema matching where schemas often reuse existing fragments of a domain or type system. Only fragments are matched and the resulting fragment mappings are combined. For instance, Saha et al. [149] stores fragments (concepts) in a repository together with mappings between them to exploit transitivity.

### 2.2.4 Instance-based Techniques

For matching two elements, their instances can be compared. Proposed techniques either rely on the overlap of instances in the source and target schema or on similarities of statistics and patterns in sets of instances.

Comparing elements based on the overlapping instances is commonly applied in existing systems. Simple approaches rely on exact matches of instances as done in QuickMig [41] and many systems that participated in the OAEI-Benchmark [54] such as RiMOM [99], Falcon [77] or AgreementMaker [31]. More advanced techniques where proposed for matching web-directories [110] or for matching ontologies with instances [91, 166]. They evaluate how strong the intersection of instance sets is, compared to the overall number of instances in the source and target. Commonly used metrics are Jaccard or Dice [82]. Recently, a first approach was described that is able to derive subtype relations between elements from comparing instance sets [28]. Only little work analyzes statistics and patterns of instance sets to compute similarities. For instance, Li and Clifton [100] extract data patterns and distributions to compute similarities of attributes in database schemas.

### 2.2.5 Graph-based Techniques

Many matching techniques were proposed that rely on structural information, i.e. the graph of elements. The underlying assumption is that similar concepts are positioned similarly in a schema and have similar relations to other elements. Often, schemas are special graphs such as directed acyclic graphs or trees. Such trees can be derived from containment or inheritance relationships or through a computation of a minimal spanning tree [170]. An advantage of trees or DAGs is that algorithms operating on trees often have lower complexity than comparable algorithms operating on general graphs. Graph-based matching techniques can be classified as global, local or taxonomy-based, which refers to the context of elements that is used for computing similarities.

**Global techniques** rely on the complete graph when computing similarities. Exact algorithms compute an exact mapping between vertices and edges of a source and target graph done by subgraph isomorphism algorithms. Inexact algorithms like graph edit distance [65], maximum common subgraph [164] or tree edit distance [181] may only find matches in the target schema for some source elements.

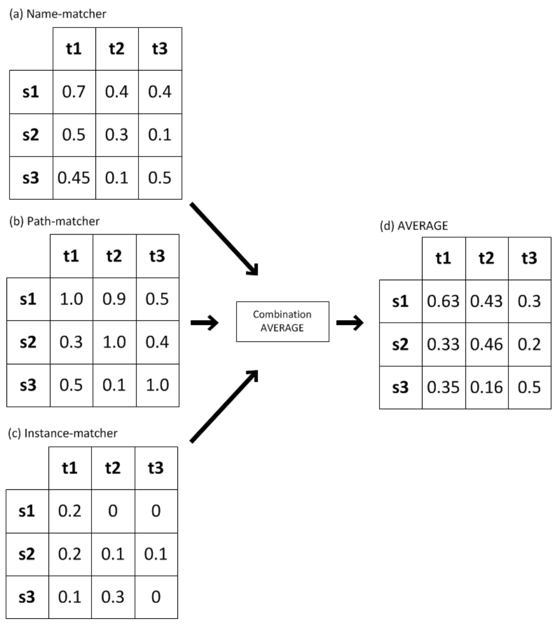
**Local techniques** solely rely on neighboring elements when computing the similarity of two elements. Often used techniques are Children, Path (Tokenpath), Leaf, Parent or the Sibling matcher that rely on the direct context of elements in a tree. For instance, the Children matcher takes as input the pair wise similarities of all children elements of the source and target element to be compared. The resulting children-similarities are combined to a single similiarty value by computing the average. More advanced propagation-based techniques compute similarities iteratively as done by Similarity Flooding [118, 117] or in the system DIKE [129]. Similarities from similar elements are propagated to neighboring elements in an iterative fashion until a fix-point or a maximal number of iterations is reached. The approach is reused in many matching systems such as GUMM [61] or within RiMOM [99].

A similar iterative technique was presented with GMO [75]. It transforms the input schemas into a bipartite graph which splits vertices into disjoint sets. Additional edges connect elements of each set and represent the similarity. Similarities of two elements are updated iteratively by combining similarities to all adjacent elements in the source and target graph.

**Taxonomy-based** techniques form a special class of graph-based matching techniques. They primarily rely on taxonomic relations i.e. the specialization hierarchy to perform the structural matching. Dedicated taxonomic matching techniques, that exploit such hierarchies where proposed with Leacock-Chodorow [97] or Wu-Palmer [175].

### 2.2.6 Logic-based Techniques

Logic-based techniques rely on a formal logical model to describe elements, their relations and additional constraints. The assumption is that two elements are similar if they share the same logical interpretations. Based on logical expressions, reasoning can be performed to derive new mappings. In most cases these techniques are used in a post processing step when a mapping was already computed. Logic-based techniques are not restricted to ontologies. Also in meta models or database schemas formal constraints can be defined for example by using the Object Constraint Language



(a) Name-matcher

| | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 0.7 | 0.4 | 0.4 |
| s2 | 0.5 | 0.3 | 0.1 |
| s3 | 0.45 | 0.1 | 0.5 |

(b) Path-matcher

| | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 1.0 | 0.9 | 0.5 |
| s2 | 0.3 | 1.0 | 0.4 |
| s3 | 0.5 | 0.1 | 1.0 |

(c) Instance-matcher

| | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 0.2 | 0 | 0 |
| s2 | 0.2 | 0.1 | 0.1 |
| s3 | 0.1 | 0.3 | 0 |

(d) AVERAGE

| | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 0.63 | 0.43 | 0.3 |
| s2 | 0.33 | 0.46 | 0.2 |
| s3 | 0.35 | 0.16 | 0.5 |

Figure 2.5: Example combination of three input mappings

(OCL) [14]. These constraints can then be used to reason about similarities of elements.

## 2.3 Combination of Matcher Results

In most matching systems the results of a set of matchers (also sometimes called matcher ensembles [107]) are combined to improve quality. As defined above, a combination operation computes a single mapping result from a number of result mappings. In Figure 2.5 an exemplary combination of three input mappings from a Name matcher(a), Path matcher(b) and Instance matcher(c) is combined to a single similarity matrix(d).

A multitude of techniques can be found in literature. In contrast to matching techniques, classifications and overviews of combination strategies [57] [137] are still incomplete. Euzenat and Shvaiko [57] distinguish three operator classes which are triangular norms, minkovski distances and weighted adjusted sum. However, additional classes can be identified that are based on heuristics such as majority voting or that are based on learning. Learning-based approaches are introduced in detail within Chapter 3 when approaches for automatic matching system configuration are discussed.

### 2.3.1 Triangular Norms & Minkovski Distances

Triangular Norms (T-Norms) are used for combining values of several dimensions so that they can also be applied for combining mappings [57]. Well known representatives of T-Norms are MIN, MAX and WEIGHTED PRODUCT. MIN always chooses the minimum value of a set of values that were computed by different matchers. This approach is very pessimistic, since it requires all matchers to return high similarity values for a pair to later "survive" a selection. MAX in comparison behaves very optimistically since only one matcher needs to return a high similarity value, no matter what other matchers compute. WEIGHTED PRODUCT is often used in multi-criteria decision making by building a product of similarity values. The drawback is that with increasing number of matchers the combined similarity values get very small.

In [57] an alternative combination method is presented with Minkovski Distances that better balance individual dimensions and do not have the mentioned problems of the WEIGHTED PRODUCT. Instead of distances, k similarity values $sim_k\left(s,t\right)$ are combined so that the the adapted Minkovsky Distance can be defined as follows:

$$mink\left(s,t\right)=\sqrt[p]{\sum_{k=1}^{K}\left(sim_k\left(s,t\right)\right)^p} \tag{2.2}$$

Minkovski Distance is a generalization of popular distance metrics like MANHATTAN Distance with $p=1$ and the EUCLIDEAN Distance with $p=2$.

### 2.3.2 Weighted Combination

Most combination approaches rely on a special case of the weighted adjusted sum of input similarities [50]. The weighted adjusted sum is defined as follows:

$$combine_{weighted-adjusted}\left(s,t\right)=\frac{\sum_{k=1\ldots n} w_k\cdot adj_k\left(sim_k\left(s,t\right)\right)}{\sum_{k=1\ldots n} w_k} \tag{2.3}$$

with $(s,t)$ being a source and target element pair, $sim_k\left(s,t\right)$ the similarity computed by the k-th matcher, $w_k$ a weight for each individual matcher and function $adj:[0,1]\rightarrow[0,1]$ beeing an adjustment function to transform the original similarities. The adjustment function is a continuous, not necessarily diffentiable function such as a sigmoid function. The following approaches are special cases of the weighted adjusted sum. They mainly differ in the way how the weights are defined.

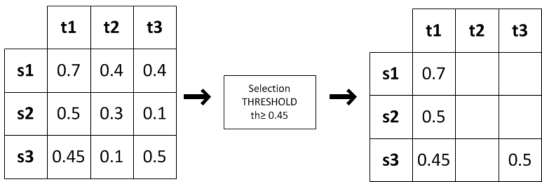
The AVERAGE strategy is the simplest version of the weighted approaches that assumes equal weights for every matcher and an identity function as adjustment function. AVERAGE is very often used by existing matching systems. It does not impose any additional parameters and often returns good results since it levels out individual weaknesses and strength of the input matchers. AVERAGE showed good results in former evaluations [36] and was also the most robust one in preliminary evaluations for this doctoral work [137].

The WEIGHTED strategy additionally sets weights for each input matcher result and is often used instead of AVERAGE. In weighted combinations the additional definition of the weights is problematic. It can be observed, that by setting the wrong weights the result quality decreases significantly more than it could increase by setting the right weights [137]. Most system designers fix those weights by manual testing or from experience. Manual definition of weights can also be supported by given matcher credibilities [168]. Often, machine learning techniques are used [51, 50, 109] to automate the definition of these weights. Since the machine learning assumes to have correct mappings as training data these approaches can often not be used. Gold standards are rare and the learned parameters could then be invalid due to the differences of the training schemas and the schemas to be matched. This happens since also weaknesses of matchers might get overweighted. In Section 3.2.1 machine learning approaches are reviewed in more detail.

Special combination approaches find alternatives to define weights as done in the OPENII-System [152]. The approach uses the absolute value of a so-called voter score as a weight to compute a confidence score. Voter scores are values between $[-1, 1]$ and are similar to similarity values. Values higher $0$ get higher confidence and values lower $0$ get lower confidence. Voter scores can easily be transformed to similarity values. The NON-LINEAR strategy from Algergawy [7] extends the weighted sum approach to include interdependencies of similarity measures into the combination of similarities for different matchers. The problem with the NON-LINEAR strategy is, that there are additional parameters introduced that are hard to set manually. Moreover, this additional effort must pay off in terms of quality and robustness in order to be a valuable alternative. The OWA (Ordered Weighted Combination) strategy [177, 88] tries to adaptively assign different weights for each pair of matched elements based on their similarity values. It automates the process of determining weights of individual matchers. The n similarity values for each compared element pair are treated as a list of similarity values. The list is ordered and each position gets a weight assigned. The weights are computed with fuzzy linguistic quantifiers (see [177] for details). Similarity values computed by a matcher may have different positions and weights for different element comparisons. According to the evaluations from [88] the At Least Half (ALH) and the MOST quantifiers performed best on ontology matching tasks. ALH only sets weights to the top half of similarity values ensuring that only these similarity values are combined. OWA MOST underweights the highest and the lowest similarity values but takes all others into account.

### 2.3.3 Majority Voting

Voting mechanism recently gained interest [74, 49]. They can be used to combine the results of multiple matchers. Eckert et al. [49] observed that a majority voting on the result of a set of matchers is a valuable combination technique. So called matcher votes indicate what percentage of a set of matchers found correspondences. Unfortunately, details about how the resulting similarity value of the combined



|  | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 0.7 | 0.4 | 0.4 |
| s2 | 0.5 | 0.3 | 0.1 |
| s3 | 0.45 | 0.1 | 0.5 |

|  | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 0.7 |  |  |
| s2 | 0.5 |  |  |
| s3 | 0.45 |  | 0.5 |

Figure 2.6: Selection example

correspondence can be computed are missing in their work. Similar observations were made in the area of large scale schema matching in the web where voting mechanisms were proposed to combine results of multiple schema matchers [74].

## 2.4 Selection of Correspondences

The presented combination techniques create a single mapping from a set of input matcher results. The result of the mapping combination still contains up to $|S| \times |T|$ correspondences. From these correspondences the most probable ones need to be selected. In Figure 2.6 an exemplary similarity matrix is selected.

A number of strategies were described by Melnik et al. [117] as well as Do et. al [35] for schema matching. Meilicke and Stuckenschmidt [114] later added strategies for ontology matching. In addition, Marie and Gal [108] gave an overview to stable marriage based selection techniques. Recall from the definition above that selection extracts the most probable correspondences from a mapping $M$ using some selection strategy. In the following, four general classes of selection strategies are described, that are threshold-based, maximum-based, optimization-based and schema-dependent strategies.

### 2.4.1 Threshold-based Strategies

The simplest selection strategies used in schema matching are threshold-based. Given a threshold $th$ in the interval [0,1], the threshold selection function $select_{thres}$ extracts correspondences from a mapping $M$ as follows:

$$select_{thres}\,(M, th) = \{(s,t,v) \in M \mid v \geq th\} \tag{2.4}$$

The threshold $th$ in equation 2.4 above is often an absolute threshold. However, the threshold could also be given as a delta value relative to the correspondence with highest similarity value, or as a proportional threshold that is defined as the percentage of highest similarity values. The problem with threshold-based strategies is that they are static for all correspondences of a schema. High absolute thresholds typically increase precision whereas lower thresholds decrease precision. The values of an optimal threshold strongly depend on the similarities of the input mappings.

### 2.4.2 Maximum-based Strategies

To solve the problems of threshold-based strategies, maximum-based strategies where proposed. It turned out that a similarity value should be interpreted relative to the value of other correspondences. In particular the relative value compared to the correspondence with maximal similarity is crucial. Popular maximum-based strategies are MAX-N and MAX-DELTA that were proposed by Do and Rahm [35] and Melnik et al. with different names [117].

MAX-N was recently also called "Dominants Algorithm" [108]. It extracts the maximum N correspondences for each source element (or alternatively target element) from a mapping $M$. The maximum N correspondences are collected differently depending on the so-called selection direction as proposed by Do and Rahm [35]. Collecting maximum correspondences for the source elements is called *forward* selection (fwd) and for the target elements it is called *backward* selection (bwd). Based on the top$N$ match candidates for a source or target element $a$ in a mapping M the forward selection can be defined as:

$$select_{maxN}\,(M, N, fwd) = \{(s,t,v) \in M \mid\ (s,t,v) \in topN(s, M, N)\} \tag{2.5}$$

Analogously, the MAX-N selection in backward direction can be defined. Note that the result of $topN$ is a set that can contain more than N values if there are correspondences with equal similarity value in the set of match candidates. The results of forward and backward direction can be intersected which is called *both*. MAX-N with $N = 1$ and direction both do not ensure that [0,1][0,1]-mappings are created which is sometimes desired.

MAX-DELTA differs from MAX-N in that it does not rely on a fixed N-value but relies on a delta value [35]. MAX-DELTA takes the MAX-N ($N = 1$) correspondences and includes correspondences that are within a delta-environment of the maximal correspondence. The size of the delta environment depends on the value of the maximal element for each row or column respectively. MAX-DELTA relaxes the strict specification of the number of match candidates an element can have in a result mapping. MAX-DELTA also proved to be the most robust selection strategy in COMA++ [11]. With the MAX-DELTA and bigger delta-values often [0,*][0,*]-mappings are created. A Similar strategy was proposed by Melnik et al. [117] that they called *Perfectionist Egalitarian Polygamy* (EGALITARIAN).

It is sometimes desired that every element is only contained in one or no correspondence which leads to a [0,1][0,1]-mapping. In order to restrict the selection result to such a cardinality Melnik et al. also proposed the EXACT strategy [118]. It restricts a selection result to only [0,1][0,1]-mappings. Certainly this selection should only be used in combination with other selection techniques since for a non-selected mapping consisting of $|S| * |T|$ correspondences the resulting mapping of the EXACT selection would be empty. In addition to that also a N-N selection can be defined that tries to restrict a result to an [0,N][0,N] mapping. This is sometimes useful if multi-mappings are expected as result.

### 2.4.3 Optimization-based Strategies

Optimization-based strategies are very different to the selection techniques described above. They transform the mapping problem to a graph matching problem so that graph matching algorithms can be applied. Most of these algorithms strive for a [0,1][0,1]-mapping using some optimization criteria [101]. Popular representatives are STABLE MARRIAGE and ASSIGNMENT strategies. Discussions and comparisons of both can be found in [108].

STABLE MARRIAGE is often used - also by RiMOM [99] and Falcon [79] in the OAEI contest. It relies on the stable matching (SM) between two sets of elements given a set of preferences for each element [71]. A stable matching is a complete one-to-one matching of two sets of elements (originally man and women) with the property that there a no two couples $(x, y)$ and $(x', y')$ such that $x$ prefers $y'$ over $y$ and $y'$ prefers $x$ over $x'$. Gale and Shapkley prove that it is possible to make all marriages stable if $|S| = |T|$ [64]. Meilicke et al. [114] as well as Marie and Gal [108] describe an algorithm called NaiveDescending/Royal Couples to compute the stable marriage. Stable Marriage strives for a local optimum that tries to make all marriages locally stable. However, striving for the local optimum could sacrifice quality. Some pairs, even with low similarity are only included because they follow the stable marriage property. Stable mappings are [0,1][0,1]-mappings so that every source element finds only one or no partner.

ASSIGNMENT strategies like Maximum Weighted Bipartite Graph Matching (MWBM) strive for a global optimum. MWBM maximizes the sum of all similarity values and ensures that a [0,1][0,1]-mapping is created. The ASSIGNMENT problem can be solved in polynomial time by using the Hungarian algorithm [94]. In an assignment not every partner is finally in its most optimal coupling. Individual partners may be less happy so that the result may not be a stable matching. Similar to the STABLE MARRIAGE the maximization of similarity values could lead to quality losses since optimizing the total sum of similarities not necessarily improves quality. In order to achieve the maximal sum, correspondences with low similarity are sometimes incorrectly included. In particular the restriction to [0,1][0,1]-mappings often leads to such situations.

### 2.4.4 Selection with Background Knowledge

There are a number of matching systems that rely on the schema information and additional background knowledge to correct correspondences of a given mapping. These selection approaches are mostly domain dependent. For instance, Meilicke et al. discuss a number of approaches for ontology matching [114]. They rely on the rules and relations of the source and target ontology to rule out wrong matches in a mapping. For instance, a mapping could break the consistency of the source and target ontology and make concepts unsatisfiable. Reasoning can then be used to discover and repair such logical inconsistencies [113] which was evaluated in [115]. For example, in the ASMOV ontology matching system [86] a number of domain specific heuristics are applied to correct and validate output mappings. Among others they remove multi-mappings (they call multiple entity correspondence) which is an assumption on the result being a [0,1][0,1]-mapping. They also remove crisscross correspondences where two correspondences are in conflict so that a child matches to a parent and vice versa. Disjointness contradictions occur if concepts in one ontology are disjoint but are in subsumption hierarchy in the other ontology. In such cases the correspondence is most probably incorrect. Other domain specific heuristics were proposed for filtering and rewriting links of generated transformations in model matching [34]. And also for matching relational schemas, correction rules are used to correct generated mappings [26]. Unfortunately, such selection strategies cannot be generically used since the rules differ strongly for each problem domain.

### 2.4.5 Combining Selection Strategies

In many systems multiple selection approaches are hybridly combined or are applied in sequence. That means the output of one selection serves as the input to another selection. Often, the THRESHOLD selection is combined with others such as MAX-DELTA or MAX-N [35]. This is reasonable, since low valued similarity values can be pruned out before selecting the maximum values. This ensures that only values with a minimal similarity can be contained in the result of the final selection. The EXACT selection is also combined with other selection approaches as done in [118] to create a [0,1][0,1]-mapping. Finally, selections that rely on background knowledge are often combined with other selection strategies often within a post processing step.

## 2.5 Comparing Combination and Selection Strategies

It already got obvious, that the number of matching techniques as well as different selection and combination strategies is high. To simplify the decision what approaches to apply for a given problem a number of evaluations were performed. Matchers or complete matching systems were already compared extensively [56, 36, 54]. In contrast to that, evaluations of selection and combination strategies are still rare. Do [36] evaluated THRESHOLD, MAX-N and MAX-Delta in his doctoral thesis. Euzenat et al. [57], Meilicke et al. [114], Melnik et al. [117] as well as Marie and Gal [108] gave introductions to combination and selection techniques but evaluations were either not existing or restricted to only few strategies and the used data sets were rather simple.

In the context of this doctoral work a broader evaluation of combination strategies [137] and also selection strategies was performed on heterogeneous sets of mapping problems. The findings are summarized briefly. They serve as motivation for further work on process-based schema matching and adaptivity in schema matching.

In the evaluations three different data sets were taken, that consist of 10 smaller mapping problems from the COMA++ Evaluation [11], from an SAP Enterprise

Service Repository (ES) and from the OAEI Benchmark 2010. As quality measure the commonly used precision, recall and f-measure were used.

In Figure 2.7 and 2.8 the result of comparing the set of implemented selection and combination operators is visualized. Average values are computed over all 30 mapping tasks.

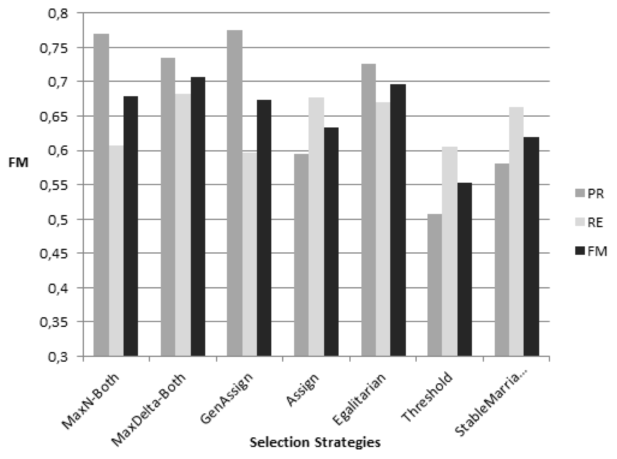


Figure 2.7: Comparing selection strategies

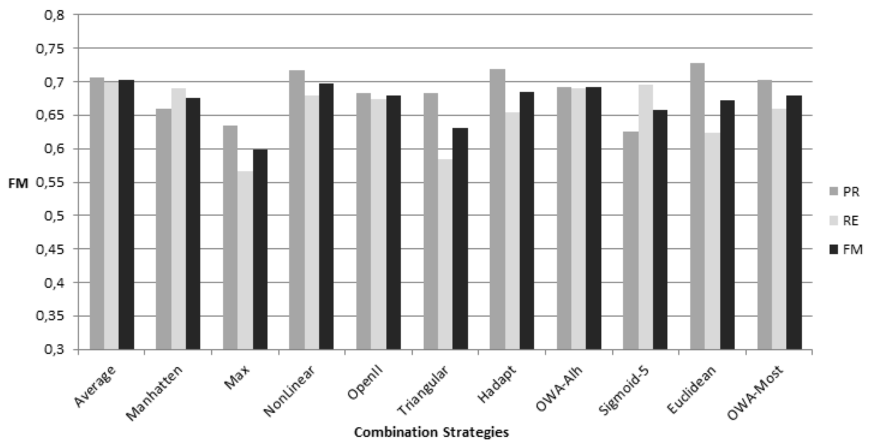


Figure 2.8: Comparing combination strategies

The results show that the choice of the most appropriate selection or combination approach highly influences the result quality. The most robust selection strategies were MAX-DELTA and MAX-N from COMA++ together with the EGALITARIAN strategy from [117]. Most other strategies have problems to compete. The Figure 2.8 illustrates that there are huge differences between the quality of the given mapping combination techniques. As was already shown in [137] the AVERAGE combination

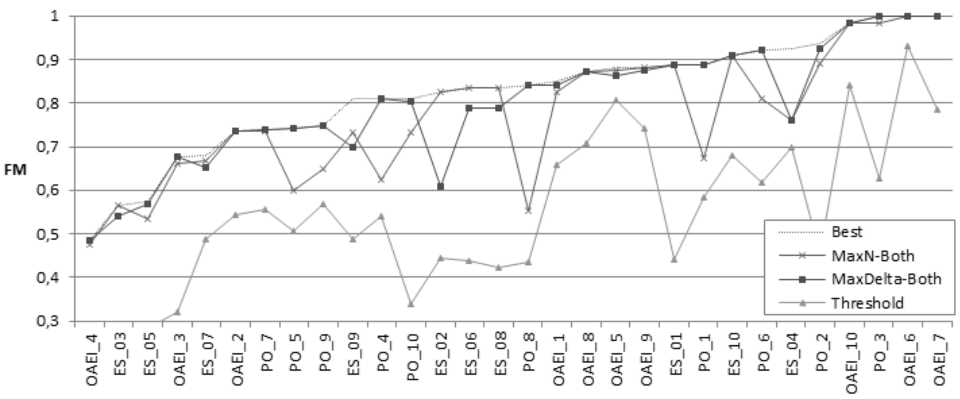


Figure 2.9: Comparing selection strategies to the best possible strategy

performs best together with NON-LINEAR [7]. But also the OWA strategy was able to perform a robust combination of similarity values.

In the evaluations from [137] also the individual performance of strategies per mapping problem was analyzed. Figures were created that contain series for each strategy and one series that captures the maximum possible result for each mapping problem (see Figure 2.9 and 2.10). The X-Axis enumerates the mapping test cases and orders them by achieved quality. On the Y-Axis the f-measure is used. The figures illustrate that there is some deviation from the maximum possible result for each of the strategies. No strategy is able to achieve the best possible quality in all cases. For instance, in some cases MAX-N performs better than MAX-DELTA. For combination strategies, AVERAGE is often very close to the optimum with rare outliers. Thus, AVERAGE is a very robust combination technique. It was surprising to observe, that choosing the wrong selection strategy from a set of good performing ones can have a stronger effect on the result quality. Therefore it would be desirable to automatically identify those cases where a specific selection (or combination) strategy is better suited.

Finally, in the influence of the number of matchers in a combination was analyzed (see Figure 2.11). The figure shows for each combination strategy the maximum achieved quality for a combination of matchers from 1 to 8 matchers on the given data set. The figure illustrates that with increasing number of matchers the result first increases but with some combination strategies also decreases if more than 5 matchers are involved. Obviously, some combination strategies are better at combining bigger sets of matchers than others. With increasing number of matchers the probability that a matcher returns 0 as similarity value in one dimension leads to problems with the product of values. Interestingly, the OWA strategy seems to be able to better cope with 8 matchers. The reason is that OWA prunes results of the lowest and highest similarities and thus reduces the number of similarity values to combine. It seems that matchers that cannot contribute to improve the mapping quality do reduce the quality.

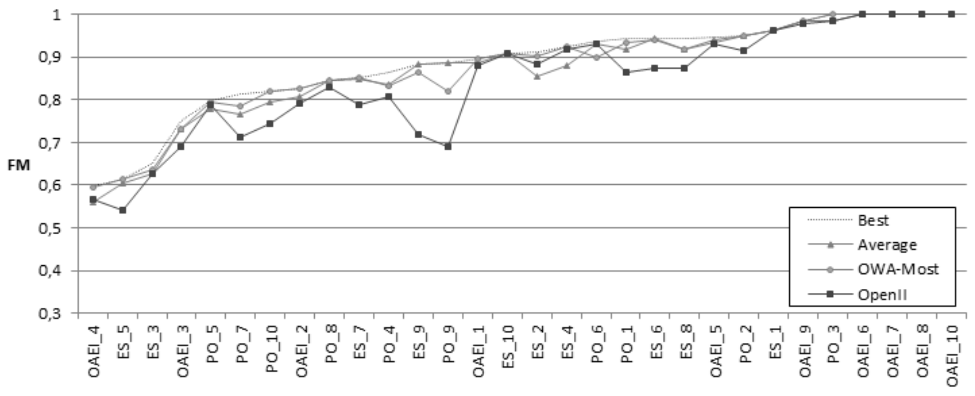


Figure 2.10: Comparing combination strategies to best possible combination strategy

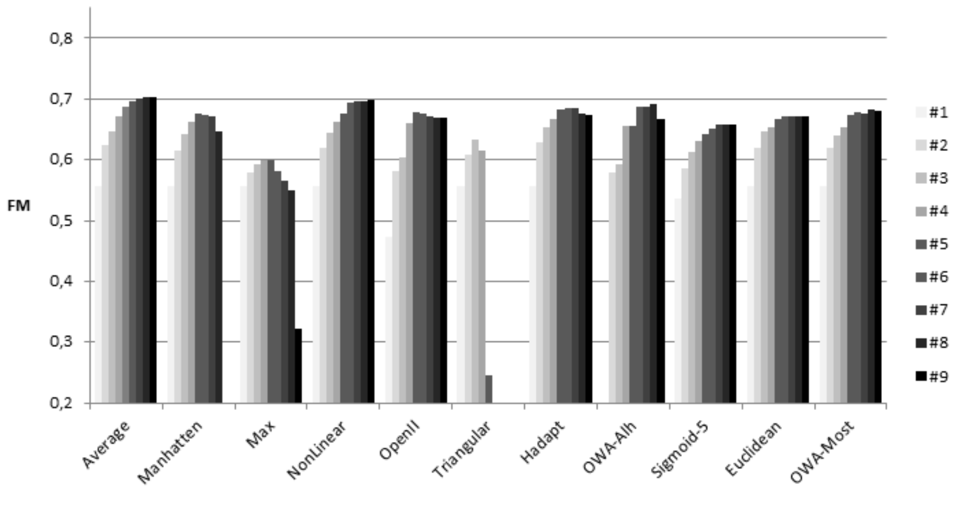


Figure 2.11: Comparison for ES mapping set

### 2.5.1 Conclusions

From these pre-evaluations a number of observations can be made. First, there are huge differences in the quality and robustness of the existing strategies. Most robust strategies are the AVERAGE combination and the MAX-DELTA selection. However, there is no combination and selection technique that is able to be better in all use cases. Automatically identifying weights as proposed by OWA and others is problematic and only in some rare cases does improve quality.

It can be observed that with increasing number of matchers the quality of combinations decreases as could already be shown by Do [36]. Obviously, 5 matchers seem to be a good choice. However, if one would like to include more than these 5 matchers then alternatives to the standard parallel combination of matchers would be needed. For instance, structural matchers could be combined separately from element-based matchers and finally results could be combined again. In order to achieve that, more flexibility in describing the order of matchers, selection and combinations would be needed. Moreover, finding parameters of selection operators automatically like the threshold of the THRESHOLD selection or the delta of MAX-DELTA is problematic. In related work on selection techniques some groups combine different selection strategies often implemented as hybrid additional selection strategy. However, a flexible combination of different selection and combination strategies is not yet possible. What would be helpful is to automatically identify from the input schemas and input mappings what strategy of selection and combination would fit best. This requires analyzing the input schemas or intermediate mapping results to support the choice of strategies or to change the execution order of matchers.

## 2.6 Strategies used in this Thesis

In the remainder of this thesis only a subset of generic selection and combination strategies as well as matchers is relevant. In Table 2.1 the most important selection and combination strategies are listed and shortly described. In Table 2.2 the most relevant matchers are listed. Most of the matchers, combination and selection strategies were already used by COMA [35] as standard strategies. The OWA-combination strategies were added since they performed well in the pre-evaluations of this thesis.

| **Operation** | **Strategy** | **Description** |
|---|---|---|
| Selection | THRESHOLD | It extracts element pairs that have similarity values greater than a given threshold. This creates [0,*][0,*]-mappings. |
| | MAX-N | It extracts N best target elements for each source element intersected with the N best target elements for each source element (both-strategy). It results in a [0,*][0,*]-mapping. For N=1, most correspondences are in an 1:1 relationship. |
| | EXACT | It extracts those element pairs that are only similar with a single partner. It results in a [0,1][0,1]-mapping. |
| | MAX-DELTA | It extracts the N best target elements for each source plus all elements that are within a delta environment. This is intersected with the N best source elements for each target plus all elements that are within a delta environment (both-strategy). This results in a [0,*][0,*]-mapping. |
| Combination | AVERAGE | It computes the average similarity from all input similarities for a given pair of elements. |
| | OWA-MOST | It orders input similarities by their value and automatically assigns weights to each position. Almost all positions get a weight $> 0$ assigned. |
| | OWA-ALH | It orders input similarities by their value and automatically assigns weights to each position. Only half of the positions get a weight $> 0$ assigned. All other similaritiy values are ignored. |

Table 2.1: Most relevant selection and combination strategies

| Matcher | Description |
|---|---|
| Name | It tokenizes the names of both input schemas and computes token similarities with the help of tri-gram, edit-distance and a synonym similarity measure that relies on a dictionary. All three similarity values and the token similarities are combined, which results in a single similarity value per element pair. |
| NameWeighted | Similar to the Name matcher the names are tokenized. However, the computed token-similarities are weighted by a TFIDF-like approach that was presented in [138] |
| Annotation | It takes the descriptions/annotations of the source and target elements, removes stop-words and applies stemming. Finally, Soft-TFIDF [29] is used to compute a similarity value. |
| Type | It computes data-type similarity based on a lookup table that stores similarity values for standard types. |
| Statistics | It computes the similarity of child and parent counts as well as the path-length. |
| Instance | All instances for a source and a target element are flattened to a string and Soft-TFIDF is applied to compute the similarity of the two instance strings. For larger instance sizes an overlap between instances is computed with the Jaccard-measure. |
| (Name)Path | The Path matcher takes as input the pair wise similarities of all elements in the paths of the source and target element to be compared. These constituent similarities are combined to a single similarity using the SimAverage combination (see COMA [35]). If no constituent similarity is given, the Name matcher is used by default to compute input similarities. |
| Token-Path | All element-names in the paths of the two elements to be compared are tokenized. For each pair of tokens a similarity value is computed with the tri-gram measure. The resulting token similarities are combined with SimAverage to compute a single result similarity value. |
| Children | The Children matcher takes as input the pair wise similarities of all children elements of the source and target element to be compared. The resulting children-similarities are combined with SimAverage to compute a single result similarity value. |
| Leaf | The Leaf matcher takes as input the pair wise similarities of all leave elements that can be reached from the source and target element to be compared. The resulting leave similarities are combined with SimAverage to compute a single result similarity value. |
| Siblings | The Sibling matcher takes as input the pair wise similarities of all elements that are siblings of the the source and target element to be compared. The resulting sibling similarities are combined with SimAverage to compute a single result similarity value. |

Table 2.2: Most relevant matchers

# Chapter 3

# Configuration of Matching Systems and Adaptivity

Up to now only the individual operations of schema matching (Matcher, Selection and Combination) were discussed. From these basic operators, complex matching systems can be built. In the last 10 to 15 years hundreds of such systems were introduced and every year new systems are presented (in the OAEI Contest from 2011 12 out of 14 participating systems where newly build [54], in 2012 7 out of 21 where new participants [55]). Many attempts were made to collect and compare systems based on some criteria such as their internal data model, types of user interfaces, schema input types, storage, cardinality of mapping results, user effort and used matching techniques to name a few [36, 18, 56]. These comparisons intend to help a user finding the most appropriate matching system for a mapping problem at hand. Still, it remains hard to rate what a strong system constitutes of. If an appropriate matching system is found, it needs to be configured to achieve good quality with an acceptable run-time performance. This configuration step is often cumbersome and in many cases involves changes on the code level.
In the following, relevant related work is presented that tries to support matching system configuration to increase quality or performance. Initially, commonly used matching system topologies are described in Section 3.1. Section 3.2 reviews work that tries to partly automate system construction and configuration. A major focus is set onto adaptive approaches that may lead to fully self-tuning matching systems. In Section 3.3 approaches are described that try to reduce the configuration effort by supporting the user. And finally, existing approaches for improving the performance of matching systems are described in detail in Section 3.4.

## 3.1 Matching System Topologies

As already discussed, currently promoted matching systems use a combination of different matching techniques for improving the quality of matching results. But,

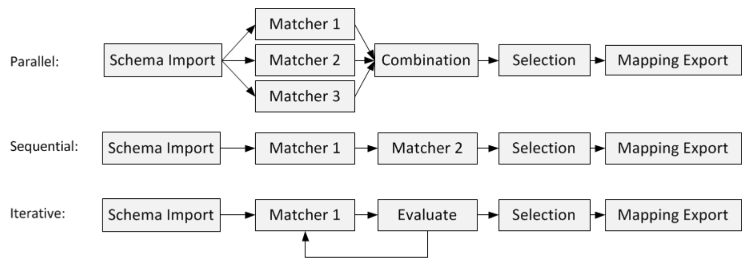


Figure 3.1: Topologies of schema matching systems

not all of them do rely on the basic matcher-combination-selection approach from above which is called parallel composition [57, 18] but also sequential and iterative processes are commonly used. Most systems are hybrid in nature [141] since they combine parallel, sequential and also iterative elements. Figure 3.1 visualizes the different topologies. Each of them is described in more detail below.

### 3.1.1 Parallel

In systems that use parallel composition matchers are executed independently which also allows for distributed execution. Results of individual matchers, often represented as similarity matrices, are combined to a single similarity matrix and a selection operator cuts off similarity values by using one of the mentioned selection strategies from Section 2.4. Typically, matchers are executed on the whole cross product of source and target schema elements which can lead to problems of run-time and memory consumption. Popular representatives are the COMA++ system [11] and AgreementMaker [31]. They allow a user to manually choose matchers, combination and selection strategies to construct arbitrary parallel schema matching processes. However, such kind of configuration support is not provided by the majority of systems. The choice of matchers and used combination and selection strategies are mostly fixed.

### 3.1.2 Sequential

Sequential combination systems rely on a sequence of matchers to narrow down a set of mapping candidates as done in CUPID [103] or within the ontology matching systems Falcon [77] and RiMOM [99]. In CUPID, first linguistic matchers are executed that are based on names and type constraints. In a second step the result is refined using a structural matching based on neighboring similar schema elements. The results of both steps are combined with a WEIGHTED combination. Also the Falcon system first executes name- and label-based matchers and then executes the structural graph matcher GMO to identify further correspondences. The order of matchers within sequential systems is mostly fixed. A step towards simplifying the construction of sequential matching strategies was done with the refine operator from

COMA++ [37]. It allows putting a mapping result as input to a matcher that can then refine the found correspondences. The performance of sequentially executing matchers is often better then parallel execution since after the first step, comparisons can be filtered.

### 3.1.3 Iterative

As discussed in Section 2.2 a number of matching approaches exist that exploit structural information. They primarily rely on heuristics that directly compute similarity values based on the similarity of the paths, parents, siblings, children or leaves [127, 35]. Some approaches extend these heuristics and iteratively change the similarity of elements based on neighboring elements. For instance, the DIKE-System [129] and Similarity Flooding [117, 118] use a fix-point computation internally to recursively compute similarity values between two input schemas. Similarity Flooding constructs a so-called propagation graph from the input schemas. That graph describes in what directions similarity values can be propagated. In each iteration, a portion of similarity of a given pair is propagated to its neighboring nodes in the propagation graph. After a couple of iterations the graph converges to a fix-point where similarity values do not change significantly with the next propagation. From the set of similarities the pairs with highest similarity are extracted by using some selection strategy. Another more recent representative is the ASMOV-System [86]. It initially computes similarity values for all pairs of source and target elements using a number of similarity measures. The results of these similarity measures are combined using the WEIGHTED combination. A pre-alignment is extracted and a semantic verification is performed which corrects correspondences and adjusts weights of input matchers based on heuristics that depend on the input ontologies. The whole process runs iteratively. Similarity values are recomputed and combined. The iteration stops when the extracted alignment does not change significantly thus a fix-point is reached. Iterative approaches often have performance issues due to the recomputation of similarity values. ASMOV took more than 3 hours to compute a result in the OAEI contest for matching ANATOMY schemas [85] whereas others completed in less than 20 minutes.

### 3.1.4 Hybrid Approaches and Workflows

Many schema matching systems mix the aforementioned topologies in their internal process. For instance, Falcon and RiMOM have both a parallel and a sequential part. RiMOM also applies Similarity Flooding as structural matcher which adds the iterative bit. Still, the internal processes of such systems are mostly fixed.

Some groups began to collect standard components of matching processes [104, 178]. The goal is to simplify the construction of matching processes to foster reuse of existing components or to treat the tuning problem separate from the chosen topology as done by Lee et al. for the eTuner systems [98]. The eTuner approach

treats a matching system as a graph of matching components with knobs that represent their parameters that can be tuned. Also with LISTOMS [183], a reference model for ontology matching systems was introduced. It separates preprocessor, dispatcher, matcher, aggregator, pruner and user interface also for tuning purposes. The dispatcher refers to a kind of controller that selects and executes the subsequent matchers, aggregators and pruners. Bernstein et al. [24] propose a system that supports a user in defining matching system behavior as so-called strategy scripts. Those strategies try to standardize components of a matching system.
Similar initiatives can be found in the entity matching field where a set of operators is used for defining match workflows [167] for matching sets of entities.

Within this thesis, a framework is introduced that also relies on a standardized set of operators (so-called matching processes) for schema matching that can be used to construct a variety of hybrid matching processes. Moreover, such processes can be graphically modeled. They are further used for automatic tuning.

## 3.2 Automating the Configuration

Obviously, the construction and configuration of a matching system is complex as it involves many different aspects like choosing the topology of the matching system, selecting matchers as well as combination and selection strategies. Each of these components then need to be parameterized and the overall run-time should also be small.

A common strategy to build a robust matching system is based on finding an optimal set of matchers and parameterizations for selection and combination. A given set of test schemas and mappings is used as reference data. A huge space of configurations is tested and the best one is chosen as default strategy as done in COMA. Such default strategies are quite robust as could be shown in a number of use cases where systems compared to COMA [66, 12, 90]. One reason is that the parallel composition itself shows to be robust. However, for new use-cases that differ strongly from the mappings that were used for building the default strategy this approach can have problems. To alleviate that, Lee et al. [98] proposed to synthetically generate a gold standard from the input schemas which can then be used to tune the system. The gold standard is generated with a set of transformation rules that perturb a given schema. However, their approach currently ignores the target schema and only the source schema is used to generate a synthetic workload (see eTuner architecture in Figure 3.2). It therefore sacrifices some of its potential.

Robust matching systems can also be built by generating multiple mappings with different system configurations. Correspondences that more often appear in generated mappings are more likely to be correct. This basic assumption is exploited by Gal et al. in their work on top-k schema matchings [107, 62] and also within the ensemble matching approach from Bin et al. [74].

Most other existing approaches either rely on (1) machine learning techniques that require correct mappings as training data or (2) they rely on the analysis of

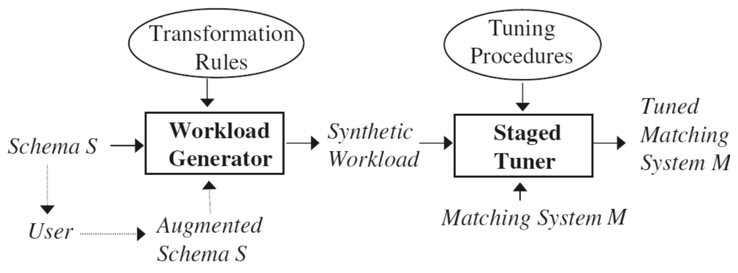


Figure 3.2: eTuner architecture [98]

input schemas and intermediate results to change parameters that can also be used to manually define rules or conditions. Each of these two classes of approaches is described in the following subsections.

### 3.2.1 Machine Learning Approaches

There is a huge body of work to support the configuration of matching systems by supervised learning techniques. A recent overview and a comparison of techniques can be found in [123]. Some systems primarily focus on learning weights for a WEIGHTED combination of different matchers as was done in APFEL [51], LSD [39] or the system presented in [19]. For instance, LSD learns weights of a so-called Meta Learner that combines Base Learners. A set of user provided mappings serves as training data. In the Meta Learner a linear regression is performed to compute weights of the WEIGHTED combination. Other systems focus on learning a classifier based on a training sample of mappings as done within Automatch [20] which trains a Bayes classifier or by Hariri et al. [13] where data mining techniques are applied. Some use decision tree classifiers [49, 2, 43] or rely on Boosting [109]. A problem of learning-based approaches is the dependency on a given set of mappings for training. The heterogeneity of mapping problems is very high so that a trained model often does not fit to a new unknown schema mapping problem.

To cope with that problem and to increase adaptivity, some first approaches include so-called schema features in the learning process. For instance, the Meta-Level Learning approach from Eckert et al. [49] measures a set of features of the input schemas like the number of concepts, schema-depth or other statistics. Those features are included when learning a decision tree. This could improve adaptivity of the learned classifier towards new mapping problems. However, even the authors acknowledge that the learned classifier still quickly overfits to the provided mapping problems. Recently, Cruz et al. [30] proposed to profile the input schemas and learn a classifier that selects the most appropriate matching system configuration from a set of predefined configurations. The configuration selection process is shown in Figure 3.3. The input ontologies are profiled, the configuration is selected and then the actual ontology matching is performed. The idea is to exploit correlations between

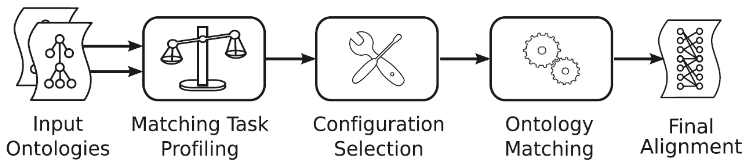


Figure 3.3: Automatic configuration selection process [30]

features and matching system configurations and to reuse fine-tuned strategies. Still, some user provided mappings are needed to train that classifier and the mapping problem must fit to one configuration in the set of predefined ones. Within YAM [46] a similar approach was proposed. It tries to help the user to select the most appropriate classifier for a mapping problem. The selection is done based on a set of input mappings, user preferences and a knowledge base of historic matching results. YAM does not rely on features or profiles.

Even though a number of groups promote learning-based solutions for schema matching, the missing reference mappings for training remains a major drawback. This is the reason, why learning-based approaches are not further investigated within this thesis.

### 3.2.2 Feature- and Rule-based Approaches

Adapting matching processes to the given input schemas can also be achieved without learning, mainly with the help of features. Do et al. stated in 2002 that the impact of features of the input and their opportunities for tuning has rarely been investigated. Since then, some attempts were made to automatically find weights of the WEIGHTED combination strategy or to define conditions and rules.

RiMOM and Falcon [99, 77] where the first to rely on features (which are also called factors) within their fixed matching process to select or unselect certain matchers. They compute Structural Similarity, Label Similarity as well as a Label Meaning to decide at run-time if a structural matcher or a Wordnet-based matcher should be executed. That helped them to better adapt to different use-cases of the OAEI-campaign and achieve better overall results. However, these conditions where fixed in the code and not generic.

Also others, like Berkovsky et al. argued that “factors such as the size of schema, application domain, and the types of schema attributes (free text, selection of pre-defined values, yes/no mark etc.) might determine the suitability of a particular matcher” [19]. They analyzed the relative performance of various matchers and the relation to given weights. Based on that, they proposed to introduce general rules to assign weights to schema matchers based on given features. An approach that implements their idea is the UFOMe system [139]. It defines a flexible workflow of matching, combination and selection supported by a graphical tool. In each phase, different modules can be used. In a so-called strategy prediction component, rules are used to select or unselect matchers based on features. Unfortunately they mainly rely on the process structure and the conditions that were used by RiMOM and Falcon. Details about when rules trigger and how they are implemented were not discussed.

The HADAPT combination [106] within the PRIOR+ system [105] was the first to analyze the intermediate mapping results while matching. Weights are automatically determined based on the so-called Harmony measure. This measure can be computed from the similarity values of the individual input mappings. The Harmony value for a mapping M between a source schema S and a target schema T is defined as

$$harmony(M) = \frac{count(M)}{min(|S|, |T|)} \tag{3.1}$$

with a function $count()$ that counts all correspondences $(s,t) \in M$ within a mapping $M$ that carry the maximum similarity for the source element $s$ and the maximum similarity for the target element $t$. It can be defined as follows:

$$\begin{aligned} count(M) = \{(s,t) \in M \,|\, \forall_{i \in T}\, sim\,(s,i) \in M :\; sim\,(s,t) \geq sim\,(s,i) \\ \wedge \forall_{j \in S}\, sim\,(j,t) \in M :\; sim\,(s,t) \geq sim\,(j,t)\} \end{aligned} \tag{3.2}$$

The resulting Harmony value is then directly taken as weight for the WEIGHTED combination:

$$combine_{weighted}\,(s,t) = \frac{\sum_{k=1\dots n} harmony(M_k) \cdot (sim_k\,(s,t))}{\sum_{k=1\dots n} harmony(M_k)} \tag{3.3}$$

Results with higher Harmony get more weight. Interestingly, Harmony showed some correlation with a computed f-measure. The problem with Harmony is, that it gives higher Harmony values to [0,1][0,1]-mappings. If a mapping does not primarily provide such one-to-one results then the weighting with Harmony could lead to problems. In the evaluations from Peukert et al. [137] the results from [106] could not be reproduced. However, the idea of the Harmony measure had a major influence on building a feature-based matching approach that is presented within this thesis. It will serve as input to a new measure that is called Monogamy in Section 4.3.3.

A second approach to automatically define weights was proposed with OWA (Ordered Weighted Combination) that was described above. It is used within the FOAM system [177, 88]. It tries to adaptively assign different weights for each pair of elements to be matched based on the output of the used matchers. OWA was originally proposed in multi-criteria decision making [177] and has also been applied to ontology matching as combination approach within [89].

An alternative work by Mochol and Jentzsch [120, 56] focusses on recommending which matching system to use for a given mapping problem. The work is relevant since the selection is based on a number of information sources like matcher metadata and schema metadata (which includes features). The information can be collected from a literature review and consulting with matching engineers. A knowledge base is built as a hierarchical tree of features that is then exploited with a formal methodology to recommend a matching system that best fits to a mapping problem. The formal process relies on predefined rules. Selecting a matching system instead of adaptively selecting or unselecting components of a system seems unpractical since a user does not want to install multiple matching systems and change them (including the GUI) for every new mapping problem. In particular the knowledge-base will likely be empty for most available and new matching systems since gathering that information is extremely cumbersome. Similar to the proposal from Mochol et al. the OntoMas mapping assistant [80] tries to help a matching system designer to build a matching system for a given use case. It also relies on a knowledge base and a set of so-called decision rules expressed as textual if-then expressions like “**If** both schemas are OWL ontologies **then** choose methods from the library that are able to match OWL ontologies”.

Another rule-based approach was recently advertised in the similarity search domain by Ryu et al. [148]. Similarity search copes with finding similar entities in a huge set of entities given an entity to search for. Ryu et al. also rely on background knowledge about usage contexts of distance functions and recommend functions based on user-given rules. With usage context they refer to the kind of entity that is matched and the attributes an entity provides. A rule consists of rule predicates like $exists()$, $valueOfLenght()$, $contains()$ that can be combined as conjunctions. An example rule looks like: $"Message" \bigwedge exist(title) \bigwedge exist(author) \rightarrow (title, EditDistance), (author, Jaro)$. It says that for matching message entities that have a title and an author attribute the EditDistance and Jaro measures should be used.

In this thesis, a novel approach is developed for schema matching that relies on features and so-called rewrite rules to construct a matching process while it is executed. The defined rewrite rules contain relevance functions that have some commonality to the predicates from Ryu et. al.

## 3.3 UI Support for Matching and Configuration

Up to now, all presented approaches mainly automated the configuration in order to simplify automatic schema matching. Another direction of schema matching research tries to bring the UI and the user interaction into the main focus when improving matching systems. This is crucial, since most existing schema matching UIs make it hard for a user to efficiently process schema matching results. Tools generate many wrong correspondences which are displayed as lines or entries in tables. Within large schemas and matching results, users loose overview and context. Moreover, UI-support for configuring and tuning a matching system is weak or not existing.

In the following, a short overview to already proposed techniques will be given. Improvements for mapping visualizations are described. Then, interactive schema matching techniques are presented and finally approaches that support the tuning and construction of matching systems are introduced.

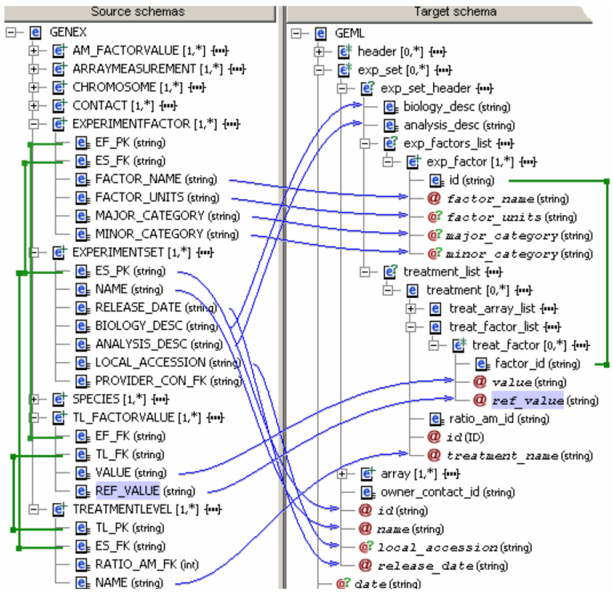


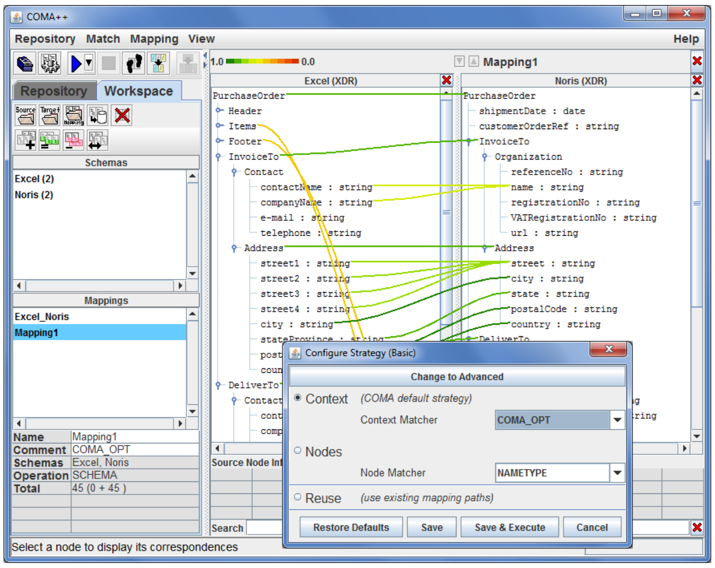
Figure 3.4: Mapping visualization from Clio [72]

### 3.3.1 Schema- and Mapping Visualization

There are already some well-known schema matching systems like COMA++, Clio or AgreementMaker with GUI support for interacting with matchers and visualizing mapping results (see mapping visualizations in Figure 3.4, 3.5 and 3.6). They mostly apply a line-based visualization that connects two schema trees.

Such visualizations have a number of problems. The number of lines can be confusing. In particular for large problems users typically have difficulties in determining what has been mapped automatically and what is left unmapped. Often, there is no difference between confirmed and unconfirmed matches. Bad matching algorithms undermine confidence in automated matching and computed similarity values are not easy to interpret by users.

Falconer et al. collected requirements for future mapping visualizations [60]. These include better schema and mapping navigation, suggestion lists of matching candidates, self-explaining results and feedback about the matching process execution. Robertson et al. [145] introduced some techniques that try to tackle some of the mentioned problems within the commercial mapping tool Biztalk Mapper (see mapping visualization in Figure 3.7). Selected or relevant mapping lines are highlighted while non-relevant lines are deemphasized. Selecting lines triggers an auto-scrolling of the source and target schema so that both the source and the target elements involved in a correspondence are visible. Trees are coalesced to reduce visible information. Matching results can be analyzed incrementally, by selecting source elements and showing suggestion lists. A filter field allows a user to only display relevant items. Lanzenberger and Sampson [95] focused on getting an overview to



| | |
|---|---|
| Name | Mapping1 |
| Comment | COMA_OPT |
| Schemas | Excel, Noris |
| Operation | SCHEMA |
| Total | 45 (0 + 45 ) |

Context (COMA default strategy)
Context Matcher: COMA_OPT
Nodes
Node Matcher: NAMETYPE
Reuse (use existing mapping paths)

Figure 3.5: Mapping visualization COMA++ [11]

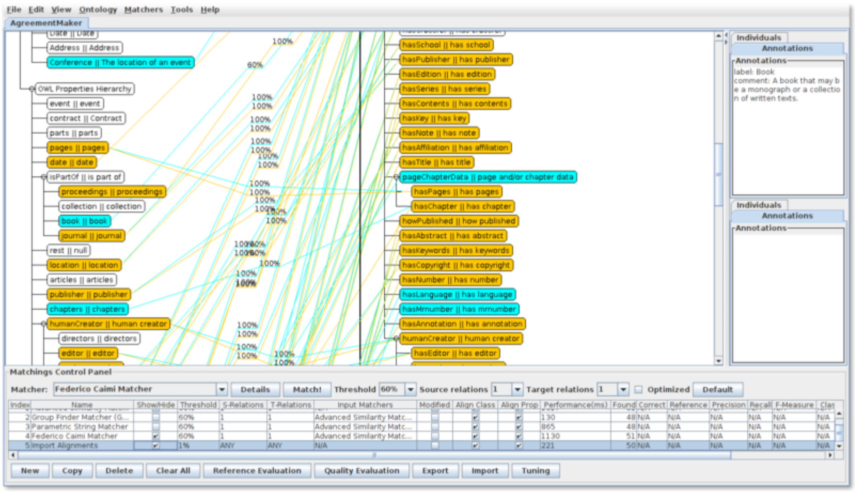


Figure 3.6: Mapping visualization from AgreementMaker [31]

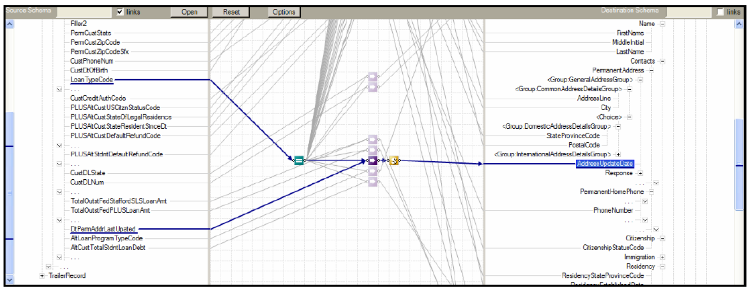


Figure 3.7: Mapping visualization from BizTalk Mapper [145]

large ontologies by relying on a cluster graph visualization that is synchronized with a tree visualization. Highlighted clusters in a mapping represent similarity (see Figure 3.8).

Falconer and Storey further propose some new techniques in their cognitive support framework for ontology matching [60]. A tree-map of the source and target ontologies as shown in Figure 3.9 helps to get an overview and to identify candidate heavy regions. The mapping line view is extended by fisheye and zooming techniques that further help the user to keep the context of elements and candidates.

Some of the presented techniques like auto-scroll, filtering, mapping suggestion lists and highlighting are also supported by the mapping visualization of SAP Netweaver CE Process Composer which is used as a mapping visualization within this thesis in Chapter 5. Further contributions to mapping visualizations are out of the scope of this thesis. However, some contribution was made by synchronizing line-based and table-based visualizations as well as suggestion lists which improves the ability of the user to navigate large-size mapping results.

## 3.3.2 Interactive Schema Matching

Different techniques were proposed that treat user feedback "as a first class citizen" [17]. They intend to better involve the user in the matching process. Existing approaches differ by the choice of candidates that are presented to the user and the way user input is further exploited to improve the matching process.

Prompt [128] was one of the first matching systems that allowed the user to correct initial sets of mapping candidates which are then used for structural matching. It also provided a first concept of candidate suggestion lists. Later, these candidate lists were extended by Bernstein et al. in their work on incremental schema matching [23]. In particular, the results of confirmed matches are used for further processing. Xue et al. [176] adjust priorities of measures based on user given information and also Duan et al. [42] exploit such user input to guide the structural match propagation in Similarity Flooding. The user input influences in which direction similarities are propagated. There is much work on the question how user input can be used to improve a learning process. As was discussed in Section 3.2.1, some systems

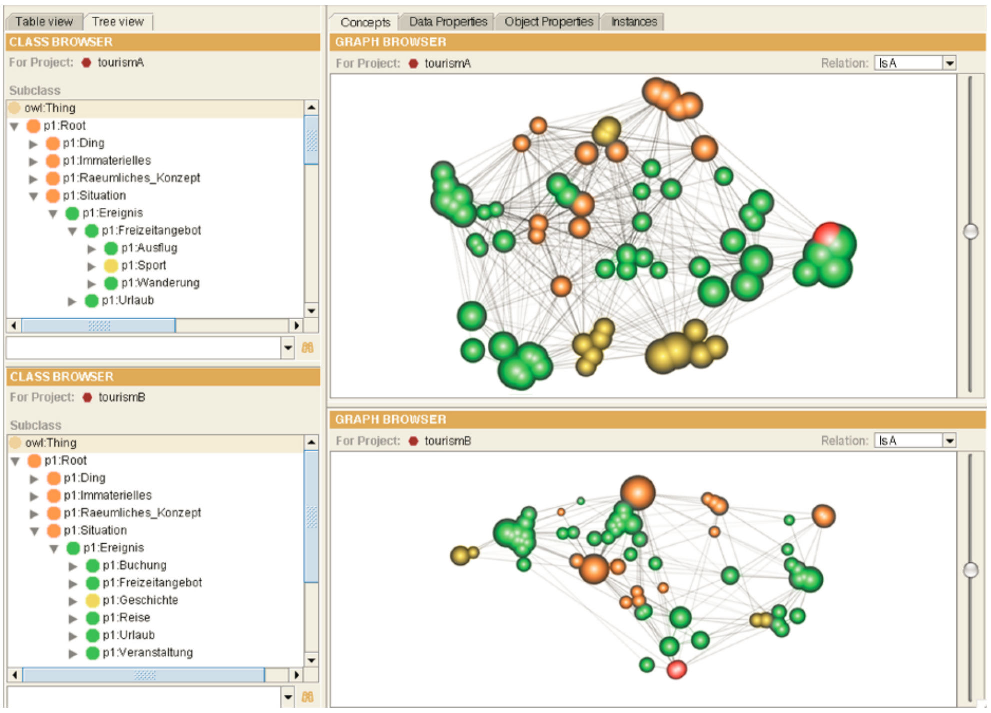


Figure 3.8: Cluster visualization from Lanzenberger and Sampson [95]

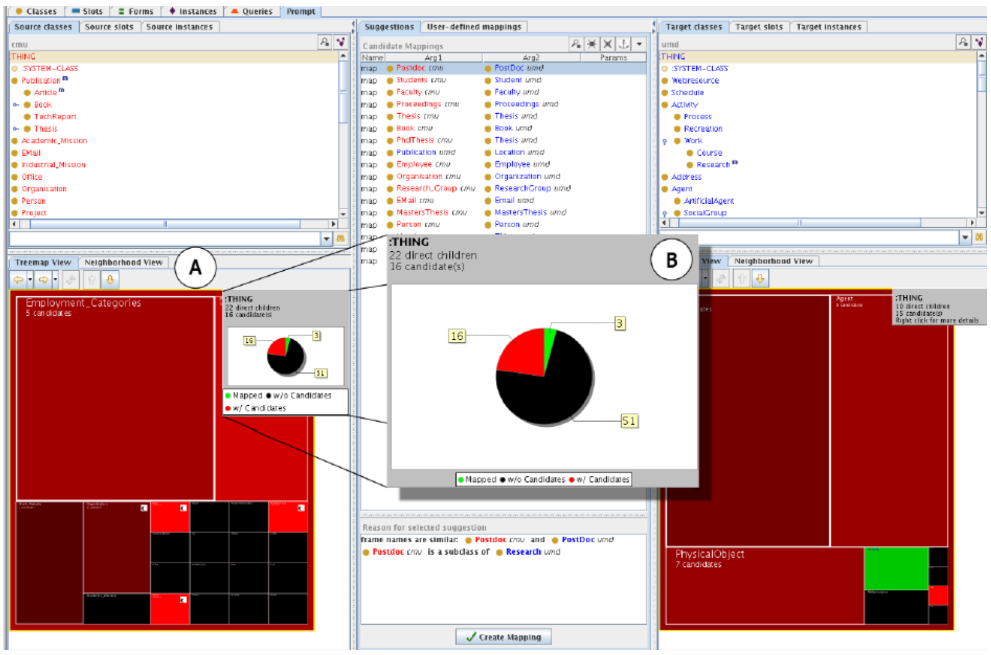


Figure 3.9: Tree-map from Falconer and Storey [60]

already perform an iterative learning process that involves the user for correcting mappings. Recent work then focusses on finding the most informative candidate mappings for user interactions [154]. The idea is to only present a selected set of correspondences to the user. Corrected matches are then again propagated by using Similarity Flooding [117]. A similar approach was proposed for pay as you go data integration systems [87] where user confirmations of candidate mappings are ordered by a criteria that measures importance. Only important candidates are presented to the user. Also Cruz et al. [32] cluster correspondences of different matchers and identify most beneficial ones that are presented to the user for correction based on a newly introduced measure they call Disagreement. The top-k Disagreement correspondences are the ones with highest variance of computed similarity values. The reduction of user input is also a topic for systems that derive mappings from provided sample instances as done by Alexe et. al.[6].

### 3.3.3 Visualization Support at Design-Time

Most of the proposed visualization techniques are primarily supporting the user when performing a mapping task at run-time. In contrast to that, design-time support for configuration and tuning of matching systems is still rare.

COMA++ was one of the first that provided an advanced interface to construct a parallel matching process by selecting, combining and parameterizing matchers. But also others subsequently provided similar features such as AgreementMaker [31]. Both tools allow creating various parallel matching processes and also support manual chaining of matcher results into a new matching process which builds trees of matchers. The overall parallel process is not visualized and cannot be changed graphically. Individual matcher development is still primarily done on the code level.

The Protoplasm system from Bernstein et al. [24] tries to support a user in defining matching system behavior as so called strategy scripts. They also introduce a graphical notation which is shown in Figure 3.10. Operators like Traverse, Filter and Match operate on the individual similarity values and similarity matrix cells which easily creates very complex configuration scripts. The UFOMe system [139] supports the visualization of a matching process on a higher level and allows a user to graphically model matching processes. The visualization represents a process as a graph of matching modules. However, the process structure is fixed to the order of matcher, combination and selection. Figure 3.11 shows the graphical user interface of UFOMe. It consists of a tab ”matching task designer” (a) that allows to construct simple matching processes. A number of so-called modules (b) with editable properties (c) can be selected and connected on a surface (d). When executing the process a logging output is provided (e).

For tuning, it can be beneficial to analyze intermediate results of matchers and matching processes. Up to now only one approach could be found in literature (except for the contribution from Peukert et al. that is presented in Chapter 5 [135]) that supports such analysis [32]. Cruz et al. [32] introduce alternatives to the line-

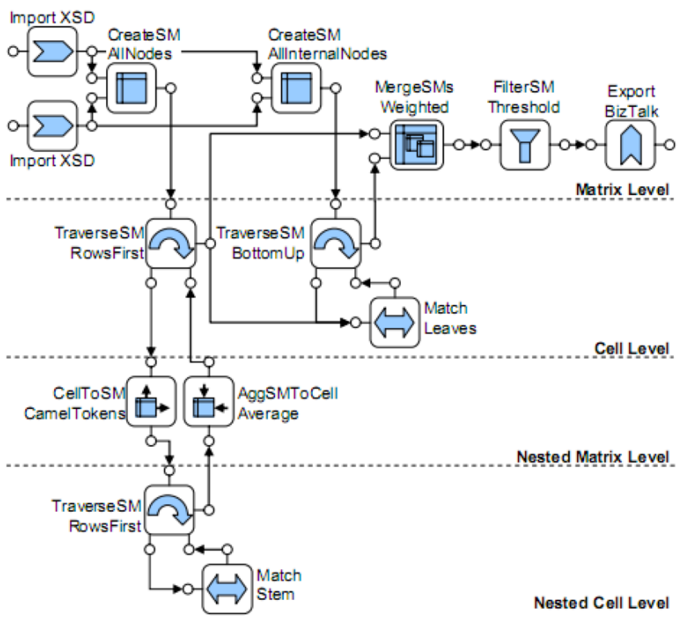
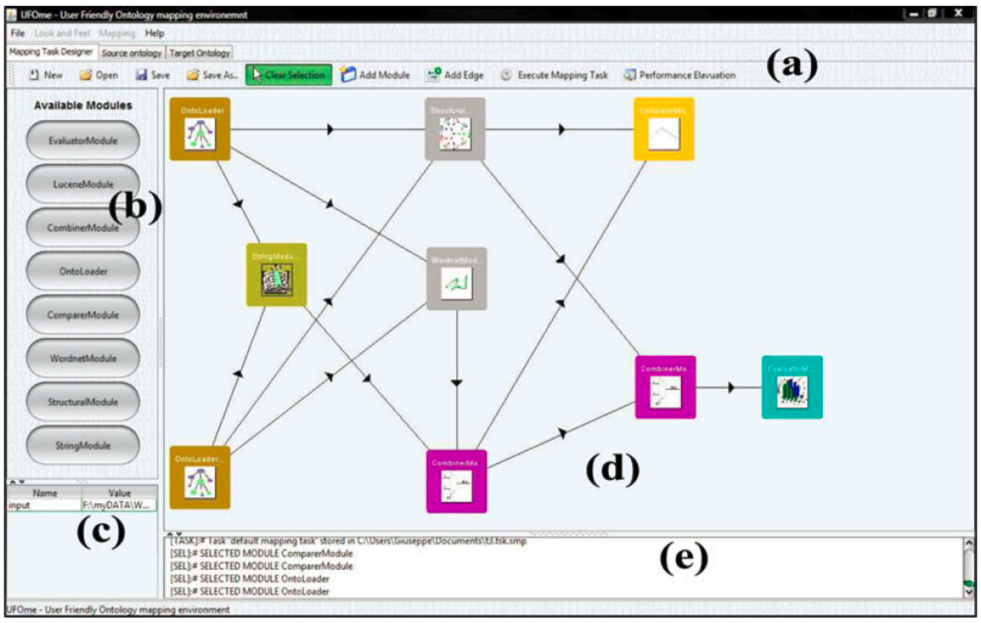


Figure 3.10: Protoplasm strategy scripts [24]



```
[TASK]:# Task 'default mapping task' stored in C:\Users\Giuseppe\Documents\t3.tsk.smp
[SEL]:# SELECTED MODULE ComparerModule
[SEL]:# SELECTED MODULE ComparerModule
[SEL]:# SELECTED MODULE OntoLoader
[SEL]:# SELECTED MODULE OntoLoader
```

Figure 3.11: UFOMe graphical matching process construction [139]

based mapping visualization for analyzing the behavior and intermediate results of matchers. Furthermore, a visual analytic panel shows similarity values of a matcher in a matrix that marks correct and incorrect matches with respect to a gold standard.

In this thesis, a major focus is set on the design time of matching processes. In Chapter 4 a process model with a standard set of operators is introduced and in Chapter 5 graphical means for constructing a matching process are presented. Tuning of matching processes is supported by forward-backward stepping and a novel cube-based visualization is described that helps to analyze intermediate results.

## 3.4 Improving Performance in Schema Matching

All presented matching techniques that involve a user need to be fast in order to not disrupt the workflow of a user. In particular for large size schemas this can be challenging. The reasons for these performance problems are obvious. Schema matching is a combinatorial problem with at least quadratic complexity w.r.t. schema sizes. Even naive algorithms can be highly inefficient on large-sized schemas. Matching two schemas of average size N using k match algorithms results in a run time complexity of $O(kN^2)$. Thus, schema matching complexity can easily explode if multiple matchers are applied on bigger-sized schemas. If an execution of a matching process takes too much time a user will be tempted to skip automatic matching altogether. Also, when tuning matching processes, fast processing is valuable. Hence, improving the performance of schema matching is as important as improving the quality. Still, only few systems have addressed the performance problem for schema matching. And most of the proposed techniques are built for individual matchers or are hard-wired within specific matching processes.

In this section existing approaches for improving the performance of matching systems and their matchers are introduced. A good overview was also given by Rahm in [140]. Existing techniques can be grouped into approaches that partition the problem (divide and conquer), filter schema parts, avoid repetitions, optimize data structures and optimize the overall process. Each of these groups is described briefly below.

### 3.4.1 Divide and Conquer

Systems that apply a divide and conquer strategy first try to manually or automatically identify relevant fragments [142, 37], blocks [78, 76], partitions [3, 131, 77] or clusters [150, 157, 156]. The further matching is then performed on these identified schema parts, which reduces the search space. Unfortunately, this approach could also worsen the overall result quality due to losses in recall. Different schema matching sub-tasks can also be executed in a distributed fashion as proposed by Gross et al. [69], Bock et.al [25] and Tenschert et al. [163].

### 3.4.2 Filtering Schema Parts

Many systems apply a schema reduction upfront by filtering out the relevant context [37] or by involving the user through a questionnaire [41]. Some automatically identify non-needed edges in the schema-graph structure [24] or apply heuristics to reduce the number of comparisons at the cost of quality [50]. Also the famous edit-distance algorithm can be improved by early pruning of comparisons [67]. Further strategies for reducing the search space were proposed in the record-linkage area. These strategies are called blocking [16] and try to reduce the number of candidate record comparison pairs while still maintaining a reasonable linkage accuracy. A technique that is particularly used within the similarity search domain is metrics-based optimization. Since the used similarity measures in that domain are often metrics the triangular inequality constraint can be exploited to filter out unnecessary comparisons [180].

### 3.4.3 Avoiding Repetitions

A generally used performance improvement technique is to avoid the repeated execution of the same subtask. For example, a pre-matching step such as tokenizing all labels avoids the repeated tokenization in later match comparisons [150]. Also Algergawy et. al. precompute so-called Prüfer Sequences that encapsulate structural relationships between elements. This saves repeated and expensive tree-traversals later in the matching process [8].

### 3.4.4 Improved Data Structures

A number of techniques use special data structures such as indexes or hash tables to improve performance. Indexing helps to quickly identify the right elements to compare with. For instance, the B-Match-Approach [44, 45] indexes tokens and its labels. That saves string comparisons based on the assumption that two similar labels share at least one common token. Others remove the nested looping effort since each element in the source needs to be compared to each element in the target by introducing a hash-join like method [24]. They also cache already computed results for later reuse.

### 3.4.5 Process-based Performance Optimization

Most performance improvement techniques mentioned so far are hard wired into fixed matching processes or act on the level of individual matchers. The topology of the matching process has a major impact on the performance and the quality of a mapping task. As already discussed, parallel combination approaches may result in performance problems if all matchers are executed for all comparisons. Sequencing the execution of matchers can improve performance since the search space is reduced by the first matchers. However, the performance improvement is achieved at the risk of losing possible mapping results. Iterative processes can have a strong impact on performance due to repeated recomputations. Up to now, no work can be found in literature that tries to construct a matching process in a way to reduce run-time. Many performance issues are only tackled indirectly. For instance, given some quality and performance requirements collected in questionnaires, some systems support the automatic selection of appropriate matchers [121] or whole matching processes out of a set of given ones [161]. Other systems like RiMOM and Falcon [99, 77] automatically select or unselect label-based or structure-based matchers depending on the specifics of the input schemas. Decision tree approaches like MatchPlanner [43] could influence performance by restricting the number of matchers and the deepness of the tree at the cost of result quality. Unfortunately, the effort of executing the matchers of a given tree for every source/target element pair is still high.

In this thesis, an approach is described that is able to automatically find the best order of matchers within a given matching process to improve run-time performance (see Chapter 6). A rule-based approach is used to optimize the performance of matching processes by rewriting it. The approach relies on filter operators that offer functionality similar to the refine operator from COMA++. With some rules, parallel combinations of matchers are automatically transformed into faster sequential combinations without changing precision and recall.

## 3.5 Comparative Summary of Matching Systems

To close the review of configuration approaches, a comparison of most influential matching systems and approaches is done in Table 3.1. It summarizes matching systems that partly automate the configuration of a matching system with respect to quality or performance or that support a user in the manual configuration task.

The table collects (1) what type of topology the internal matching process of a matching system employs, (2) whether UIs are present that support the design of the matching process and (3) if a matching process is automatically constructed or (4) automatically parameterized to increase quality.

In order to better differentiate what parts of a system are automatically tuned a difference is made between the ability to parameterize the combination, selection or to automate the choice of matchers (5). Furthermore the table collects (6) if a system relies on features of the input schemas to tune a system. The last column reflects the ability of a system to automatically tune a system towards performance.

The table lists most important schema matching systems that were already mentioned above. Cupid and COMA++ are included as well known representatives for classical matching systems that do not support automatic configuration. In Cupid, the choice of matchers as well as the choice of selection- and combination strategies is fixed. The matching process consists of parallel as well as sequential parts. There is no automatic tuning of quality or performance aspects possible. COMA++ offers a GUI that allows a matching expert to flexibly configure a parallel matching process by manually selecting and parameterizing matchers, selection- and combination

| **System** | **Topology** | **UI for design-time** | **Auto. process constr.** | **Auto. parameter tuning** | **System selects or configures Combine/Select/Matcher at run-time?** | **Feature-based** | **Auto. performance tuning** |
|---|---|---|---|---|---|---|---|
| Cupid | p & s | - | - | - | fixed/fixed/fixed | - | - |
| COMA++ | p & s | (✓) | - | - | fixed/fixed/fixed | - | - |
| OpenII | p & s | (✓) | - | - | auto/fixed/fixed | - | - |
| PRIOR+ (HADAPT) | s | - | - | (✓) | auto/fixed/fixed | ✓ | - |
| FOAM (OWA) | p & i | - | - | (✓) | auto/fixed/fixed | - | - |
| RiMOM | p & s | - | - | (✓) | fixed/fixed/auto | ✓ | - |
| Falcon | p & s | - | - | (✓) | fixed/fixed/auto | ✓ | - |
| YAM++ | p & s | - | - | (✓) weights | auto/fixed/fixed | - | - |
| APFEL | p & i | - | - | (✓) weights | auto/fixed/fixed | - | - |
| MatchPlanner | s | - | ✓ | (✓) train | fixed/fixed/auto | - | (✓) |
| YAM | n.a. | - | ✓ | (✓) train | n.a | - | (✓) |
| Meta-Level Learning | s | - | ✓ | (✓) train | fixed/fixed/auto | ✓ | (✓) |
| AgreementMaker | p & s | (✓) | - | (✓) train | fixed/fixed/fixed | ✓ | - |
| eTuner | p & s & i | - | - | (✓) synth. gold | auto/auto/fixed | (✓) | - |
| UFOMe | p & s | (✓) | - | ✓ | auto/auto/auto | ✓ | - |
| Thesis contributions | p & s & i | ✓ | ✓ | ✓ | auto/auto/auto | ✓ | ✓ |

Table 3.1: Comparing support for automatic and manual configuration of current systems

strategies. There is no support for changing and constructing the underlying matching process even though manually plugging matching results together is possible by using a refinement operator. COMA++ also offers a mapping view that allows inspecting results of matchers and computed similarities. Since further tools for analyzing intermediate results are not provided the marker is put in parenthesis. The same holds for the OpenII system which contains a matching tool with features similar to COMA+. However, within the OpenII matching system, a basic adaptive combination technique is applied that uses the similarity value of pairs as weight in a weighted combination.

Then, a group of systems is listed that introduce some kind of automatic configuration of either the choice of matchers, selection parameters or weights of the combination. These systems are PRIOR+, FOAM (OWA), RiMOM and Falcon. PRIOR+ and FOAM rely on a fixed matching process. They introduce adaptive combination techniques with the HADAPT or the OWA strategy. HADAPT can be seen as the first strategy that relies on a feature of intermediate mapping results which is called Harmony. RiMOM and Falcon also have a fixed internal matching process. They were the first systems that are able to change the used set of matchers based on features of the input schemas.

The next group consists of learning-based systems that are APFEL, MatchPlanner, YAM, Meta-Level Learning and AgreementMaker. APFEL relies on user input to train weights of the combination in an iterative fashion which introduces some adaptivity. The internal matching process which is parallel and iterative cannot be changed. The training does not rely on features of the input schemas. MatchPlanner relies on learning a decision tree. At run-time it is able to select the most appropriate matchers. By restricting the depth of the tree the performance of the process can be influenced which leads to a partial support for automatic performance tuning. YAM automatically recommends a trained classifier from a set of predefined ones based on user preferences and a knowledge-base of former matching results. Since classifiers not rely on a matching process with matcher, combination, selection some entries do not apply. YAM++ has a similar name but rather is a classical type of matching system. It relies on a process as described by Cupid with element-level matchers and a propagation-based structural matcher. What is novel about YAM++ is that is compares the results of element level and propagation-based matchers during execution and derives thresholds and weights from this comparison. Meta-Level-Learning is actually not a matching system and rather an approach. It is the first learning-based approach that relies on features of the input schemas when constructing a decision tree. Since the approach strongly relies on training data the system is not fully self-tuning. The AgreementMaker system was recently extended by an approach that learns from features what matching process to use from a set of predefined matching processes. In that respect the approach is similar to YAM that also relies on predefined processes (classifiers). However, YAM did not incorporate schema features into the automatic selection. In contrast to many other existing systems, AgreementMaker introduces advanced visualizations and UI support for analyzing matcher results. Still, the actual underlying parallel matching process is fixed as it was in COMA++ where also a reuse of former matching results was possible.

eTuner forms a separate group of systems as it does not rely on learning and does only indirectly incorporate schema features to automate the quality tuning. The goal of eTuner is to tune a given arbitrary matching process. The tuning is restricted to parameters of existing components such as combination and selection of a process. The actual choice of matchers is fixed.

Finally, rule-based approaches are described with UFOMe and the approach that is introduced within this thesis. UFOMe introduces a tool for graphically modeling a matching process. Still, the process is fixed to a sequence of matcher, combination and selection. The system tries to automate the parameterization of the selection, combination and the choice of matchers by relying on a small set of rules. The constructed process can then be executed and intermediate results can be analyzed with respect to precision/recall. Further tools for analyzing intermediate results are not provided. Moreover, performance tuning is not supported. Still, UFOMe is the work closest to the contributions that are described in the following chapters.

In this thesis, several approachs are presented that allow constructing various types of matching processes by using a graphical process visualization and a drag and drop metaphor. Furthermore techniques for intermediate result visualization are introduced that can be used for interactive tuning. Modeled matching processes can be automatically optimized to improve performance by rewriting the process structure. A novel approach for automatic process construction relies on features and rules to create matching processes specifically suited for a given mapping problem. The rules adaptively change combination and selection strategies as well as the choice of matchers by rewriting a matching process.

**Part II**

# Process-based Schema Matching

# Chapter 4

# Adaptive Matching Process Model

The review of approaches supporting the user in the matching system construction task revealed a common research direction. Many groups try to abstract from specific matching systems and introduce sets of operators and process-like descriptions of the configuration of a matching system for tuning [98], graphical modeling [139, 24] or standardization [183, 104, 178]. Such abstraction is also valuable for supporting reuse of existing matching system components or for building problem specific matching processes. Unfortunately, existing process models are either too complex like the Protoplasm strategy scripts [24] or are missing crucial operators and control structures to model adaptivity and performance of a matching system such as conditions, partitioning, filtering or looping.

In the following, a new matching process model with its operators is described. The model forms the basis for manual process modeling, automated process-based performance optimization as well as adaptive process construction.

## 4.1 Matching Process Definition

The matching process contains all steps necessary to produce a mapping from two input schemas. The basic matching process model that was introduced by Lee et al. [98] is extended within the following definitions. In particular, the idea of a comparison matrix is newly introduced to the process model.

**Definition 9.** ***(Comparison Matrix)*** *A Comparison Matrix CM defines which elements of a source schema need to be compared with elements of the target schema. CM is a matrix with $|S| \times |T|$ cells. Each cell $cm_{ij}$ contains a Boolean value that defines whether the comparison between two elements $s_i$ and $t_j$ should be performed in a following matcher operation.*

The Comparison Matrix is introduced for modeling performance aspects on the process level. Its role will be described later in detail.

**Definition 10.** ***(Matching Process)*** *A Matching Process MP is a triple (O, G, P) with O representing a set of process operators that are organized in a directed acyclic graph*

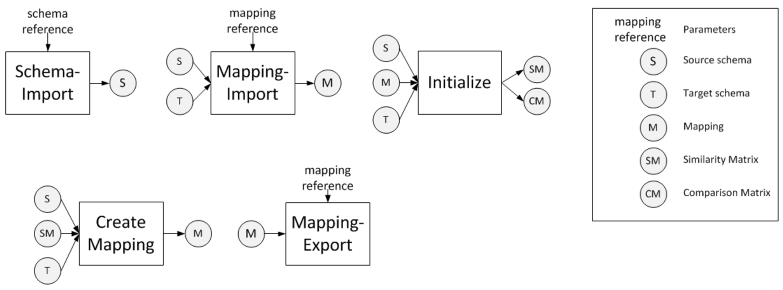


Figure 4.1: Notation for Import and Initialize operators

*G. Edges within the graph determine the execution order of operators from O and the data flow between operators (exchange of schemas, similarity matrices, and comparison matrices). P consists of a set parameters that are assigned to each operator in O. Process operators in the graph take schemas, comparison matrices or similarity matrices as input and return comparison matrices, similarity matrices or schemas as output. The output of one operator can be used as input for another one.*

The input to a schema matching process consists of two schemas $S$ and $T$ or a precomputed mapping $M$. Intermediate mapping results within the matching process are represented as similarity matrices (see definitions from Chapter 2). The output of a schema matching process is a mapping.

## 4.2 Operators of the Matching Process

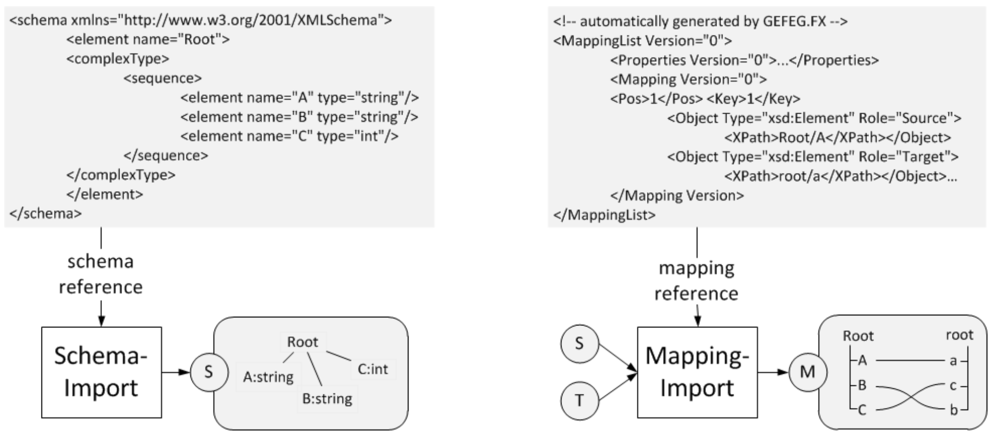
In the following, a basic set of process operators is introduced. The goal is to allow users to model the internal configuration of most existing schema matching systems explicitly as matching processes with a limited set of operators.

### 4.2.1 Import/Export Operators

The import and export operators form the interface to a matching process. The matching process contains operators for importing and exporting different metadata structures and existing mappings into process specific data formats which are schemas and similarity matrices. The notation for all such operators is shown in Figure 4.1.

#### Schema-Import

The Schema-Import operator is parameterized with a reference to a metadata structure such as an ontology, meta model, XSD schema or relational schema. It imports such structures into the generic schema representation which was introduced in Chapter 2. It consists of schema elements, structural relationships and attributes. On the left of Figure 4.2 the Schema-Import operator is shown with an exemplary XSD schema that is imported into the internal schema representation.



```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
        <element name="Root">
        <complexType>
                <sequence>
                        <element name="A" type="string"/>
                        <element name="B" type="string"/>
                        <element name="C" type="int"/>
                </sequence>
        </complexType>
        </element>
</schema>
```

```
<!-- automatically generated by GEFEG.FX -->
<MappingList Version="0">
        <Properties Version="0">...</Properties>
        <Mapping Version="0">
        <Pos>1</Pos> <Key>1</Key>
                <Object Type="xsd:Element" Role="Source">
                        <XPath>Root/A</XPath></Object>
                <Object Type="xsd:Element" Role="Target">
                        <XPath>root/a</XPath></Object>...
        </Mapping Version>
</MappingList>
```

Figure 4.2: Schema-Import operator: Importing an XSD schema into an internal schema representation (left), Mapping-Import operator: Importing GEFEG-mapping to internal mapping representation (right)

### Mapping-Import

Similar to Schema-Import, the Mapping-Import operator is parameterized with a reference to an existing mapping in some external mapping format which could be the AlignmentAPI[1], the GEFEG [2] or transformation code like ATL [3] mapping format . The output is a generic mapping that consists of a set of correspondences as was defined in Section 2.1. An exemplary import of the a GEFEG-mapping into the internal mapping format is shown on the right of Figure 4.2.

### Initialize

The Initialize operator gets two schemas and an optional mapping as input and generates a similarity matrix and a comparison matrix as output. If a mapping is given the respective entries in the similarity matrix are set to 1. Again, an example is given in the left of Figure 4.3.

### Create-Mapping

The Create-Mapping operator gets a similarity matrix as well as the source and target schema as input and generates a mapping as output (see Figure 4.3 on the right for an example). The difference to the Mapping-Import operator is that the mapping is created from a similarity matrix instead of an existing mapping file in some external format. The mapping can be further processed by post-processing operators to remove conflicting entries in a mapping or to create multi-mappings.

[1]Standard API for expression and sharing ontology alignments - http://alignapi.gforge.inria.fr/
[2]Gesellschaft fuer Elektronischen Geschaeftsverkehr - http://www.gefeg.com/de/gefeg.fx/fx_kurz.htm
[3]ATL transformation language - http://www.eclipse.org/atl/

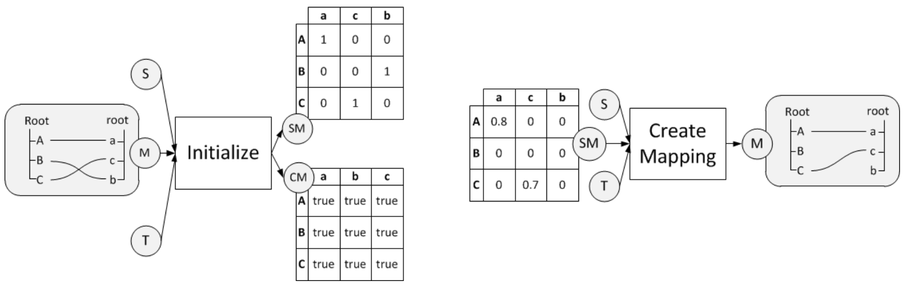


Figure 4.3: Initialize operator: Importing a schema into similarity and comparison matrix (left), Create-Mapping operator: Exemplary similarity matrix SM and mapping (right) M

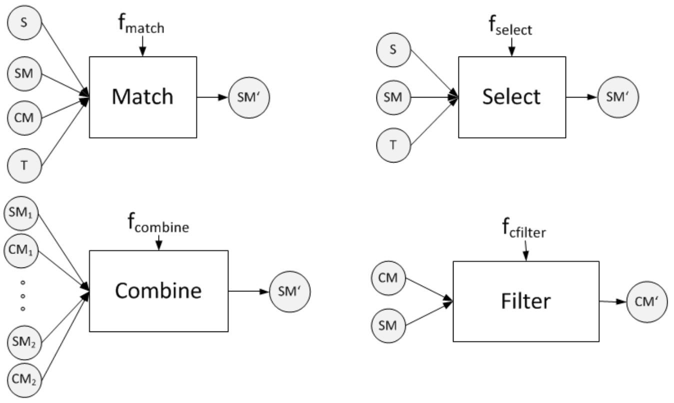


Figure 4.4: Match processing operators notation

### Mapping-Export

Finally, the Mapping-Export operator exports the final mapping into some external mapping format such as AlignmentAPI or GEFEG-mappings or transformation rules.

### 4.2.2 Match Processing Operators

The match processing operators are the most crucial operators since they perform the actual matching task that involves similarity computation, selection, combination and filtering. The match processing operators and their notation are shown in Figure 4.4

#### Match

The Match operator is defined as a function that takes a source schema $S$, a target schema $T$, an optional similarity matrix $SM$ and a comparison matrix $CM$ as input

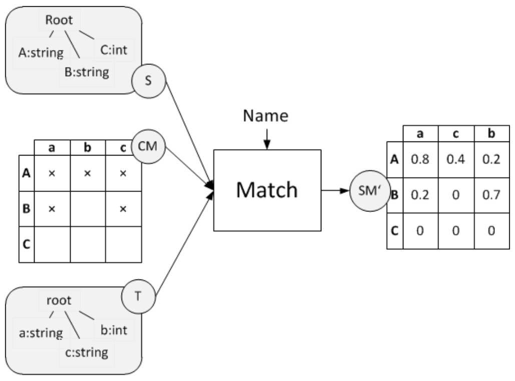


Figure 4.5: Match operator: Example comparison and similarity matrix

and computes a similarity matrix $SM'$ as output: $Match : \; S, T, SM, CM, f_{match} \rightarrow SM$' (see Figure 4.5 for an example).

It is parameterized with a matching algorithm $f_{match}$ that is executed by the Match operator. The operator only computes similarity values for element pairs where the comparison matrix entry is set to true. In addition to the schema input and comparison matrix, a similarity matrix can be provided as input to the Match operator. This can be used in particular for structural matchers to compute similarity values from a combination of precomputed similarity matrices as supported in COMA [35]. This optional mapping is also referred to as constituent similarity matrix. Other authors alternatively refer to first line and second line matchers in that context [109]. An entry $sm_{ij} \in SM'$ is computed by $f_{match}(s_i, t_j, SM)$ if $cm_{ij} = true$ with $cm_{ij} \in CM$.

## Combine

A Combine operator combines multiple similarity matrices to a single combined similarity matrix. The input matrices can result from executing Match operators but also from previous Select or Combine operators. For each pair of schema elements a combined similarity value is computed. The Combine operator is defined as a function $Combine \; : \; SM^1, \ldots, SM^n, \; CM^1, \ldots, CM^n, \; f_{combi}, P \; \rightarrow \; SM', \; CM'$. It takes $n$ similarity matrices $SM^1, \ldots, SM^n$ and $n$ comparison matrices $CM^1, \ldots, CM^n$ and a combination strategy $f_{combi}$ as input and computes a similarity matrix $SM'$ and a comparison matrix $CM'$ as output. The entries $sm_{ij} \in SM'$ are computed by calling the combination function $f_{combi}$ for each set of input values: $sm_{ij} = f_{combi}(sm^1_{i,j}, \ldots, sm^n_{i,j})$. $f_{combi}$ implements one of the mentioned combination strategies from Section 2.3. Some strategies like WEIGHTED require weights to be set for each input mapping. These can be specified as additional parameters $P$. The input comparison matrices $CM^1, \ldots, CM^n$ are combined to a single output comparison matrix $CM'$ as follows. An entry $cm_{i,j}$ is true if one of the $n$ input entries is true: $\exists cm_{a,b} \in \{cm^1_{i,j}, \ldots, cm^n_{i,j}\} \; : \; cm_{a,b} = true$. This ensures that comparisons are only filtered explicitly by Compar-

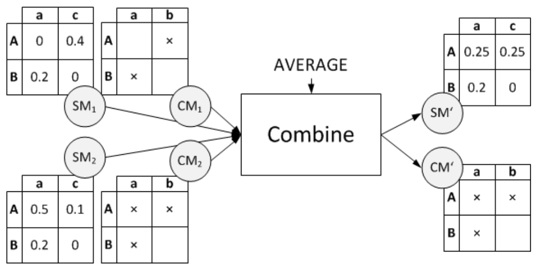


Figure 4.6: Combine operator: Example input and output

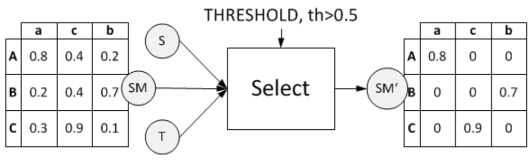


Figure 4.7: Select operator: Example THRESHOLD selection result

ison Filter operators (see below). The notation of the operator and an exemplary combination of similarity and comparison matrices is shown in Figure 4.6. In the example, the AVERAGE combination is used.

### Select

The Select operator is defined as a function $Select : S, T, SM, f_{select}, P \rightarrow SM'$. It gets a source and target schema $S$ and $T$ as well as a similarity matrix $SM$ as input and computes a similarity matrix as output. It extracts the most promising entries using some selection function $f_{select}$ that implements a selection strategy from Section 2.4. Again, a set of additional parameters $P$ can be provided. If an entry $sm_{ij} \in SM$ is not extracted then the entry $sm_{ij} \in SM'$ is set to zero. The result can be a sparse similarity matrix where similarity values are often only set for few element pairs per schema element. The notation and an exemplary selection result is shown in Figure 4.7. In the example a THRESHOLD selection is applied.

### Filter

The Filter operator is defined as a function
$Filter : SM, CM, f_{filter}, P \rightarrow CM'$. It takes as input a similarity matrix $SM$, a comparison matrix $CM$ and a filter function $f_{filter}$. Also additional parameters can be provided with $P$. The function output is a comparison matrix $CM'$. The operator filters comparisons for later executions of the Match operator by setting entries in the comparison matrix to false. The filter function $f_{filter}$ takes $SM$ as input and computes a comparison matrix $CM^{filter}$. A Boolean entry $cm'_{i,j} \in CM'$

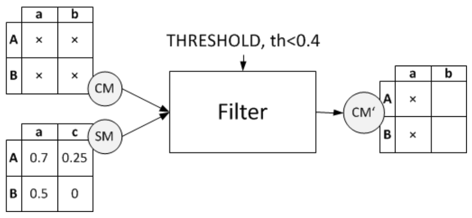


Figure 4.8: Filter operator: Exemplary threshold-based filtering

is computed from the input comparison matrix $CM$ and $CM^{filter}$ as follows: if $cm_{ij}^{filter} = true \;\wedge\; cm_{i,j} = true$ then $cm'_{i,j} = true$; otherwise $cm'_{i,j} = false$;

The difference of the Filter operator to the Select operator is, that the Filter operator computes a new comparison matrix whereas the Select operator computes a new similarity matrix. The Filter operator can be used to reduce the number of comparisons for subsequent operators in a matching process. It directly influences the flow of comparisons in a matching process to the appropriate operators. A number of filter strategies can be defined, that either rely on the input similarity matrix or the input schemas. Known strategies identify the relevant context and non-needed edges in a schema graph and remove such comparisons [37] or apply heuristics to reduce the number of comparisons, sometimes at the cost of quality [50]. More details on the Filter operator are given in Chapter 6.

### 4.2.3 Control Structures

Recently, schema matching systems introduced features to automatically adjust parameters of the combination or the choice of matchers (see Section 3.2.2). This is done to improve robustness as well as performance of a matching system. In order to be able to represent such behavior within a matching process, control structures similar to programming languages are needed that allow for conditional execution, and iteration. Furthermore, blocking approaches should be supported by an operator to improve performance of a matching process.

In the following, the Condition, Loop and For-Each constructs are described.

#### Condition

A Condition evaluates features to decide which branch to take in the subsequent matching process part. The Condition is an important novel construct to make a matching process adaptive to the matching problem at hand. Features can be computed from the input schemas as well as from intermediate similarity matrices. For example Falcon [77] and RiMOM [99] compute the structural or linguistic similarity of the input schemas to select matchers. Linguistic similarity is computed by counting the number of similar names of two schemas wheras structural similarity (structuralSim) looks substructures with similar child-counts. Both features will be

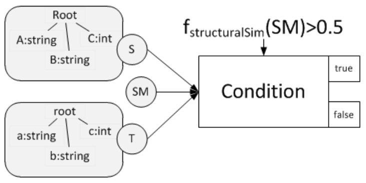


Figure 4.9: Condition construct: Example with computing structural similarity

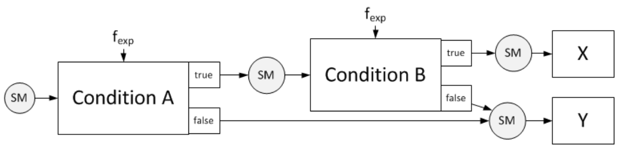


Figure 4.10: Conjunction of conditions in a matching process

discussed in more detail in Section 4.3.2 and Section A.2.3. Conditions can be used to let a process automatically adapt to the concrete mapping problem. A Condition operator gets a source and target schema, a similarity matrix and a comparison matrix as input. Furthermore, the operator is parameterized with a function $f_{exp}$ which evaluates a simple condition expression that is based on feature values. An expression consists of a feature, the mathematical operators $\leq$ and $\geq$ and a constant value c. Example feature values could be the linguistic or structural similarity of the input schemas. The Condition construct forwards the input to one of two possible outputs. If the expression $f_{exp}$ evaluates to true the input is forwarded to the positive output. If the expression evaluates to false then the input is forwarded to the negative output. Figure 4.9 shows the notation of the Condition with two example input schemas. If the structural similarity of the two input schemas is high the expression $f_{structuralSim}(S,T) > 0.5$ would evaluate to true. By chaining multiple Conditions a conjunction and disjunction of Conditions can be modeled. In Figure 4.10 the output of Condition A is input to Condition B. If the expression of A and B evaluate to true then operator X is executed: $A \wedge B \rightarrow C$. Similarly, a disjunction is modeled.

The Condition construct is needed for modeling adaptive matching processes. In Section 4.3 more details about features are given. It is important to note that the condition construct can not only be used to increase quality but also to increase performance by choosing appropriate matchers and other operators for a given schema size.

### Loop

Some matching systems execute single matchers or a whole process repeatedly until some condition is met. For instance, the fix-point computation in Similarity-Flooding repeats a propagation algorithm until a fix point is reached. The input to a Loop can be a source and target schema, a comparison matrix as well as an optional similarity matrix. The Loop takes as parameter an expression function over a computed feature

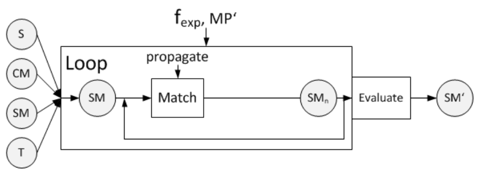


Figure 4.11: Loop construct: Notation and example

of the result of the current and previous iterations. In addition, it is parameterized with a matching process $MP'$. This process is repeatedly executed within the Loop. In the example of Figure 4.11 the matching process $MP'$ only consists of a Match operator that implements the propagation from Similarity Flooding. However, also more complex matching processes can be repeatedly executed. When the condition expression $f_{exp}$ evaluates to true the looping stops. In Similarity Flooding the euclidean distance of the current result $SM_n$ to the previous result matrix $SM_{n-1}$ should be smaller than a given $\epsilon$. The output the Loop is the result matrix of last execution of the given matching process $MP'$.

### For-Each

The For-Each construct is similar to the Loop. However, its primary goal is to reduce the size of mappings in memory. The input to the For-Each operator is a source and target schema, a similarity matrix and a comparison matrix. The operator is parameterized with a splitting function $f_{split}$ that creates small possibly intersecting mapping problems from the input problem. $f_{split}$ gets as input two schemas and a comparison matrix and an optional similarity matrix. It computes as output a number of partially empty comparison matrices. Each comparison matrix represents a smaller matching problem or block of comparisons that needs to be matched. For identifying these smaller mapping problems, blocks, clusters or partitions can be built as was discussed in Section 3.4. A well known implementation of such splitting function was presented with Falcon [77]. However, also other approaches can be used that first identify schema fragments in the source that are then only compared with few other fragments in the target (see [37]). Each of the identified blocks of comparisons is executed with the given matching process MP'. The results of matching the blocks are combined with a MAX-N (N=1) combination. In Figure 4.12 the For-Each notation is visualized. In the Figure, the mapping problem of matching $S$ with $T$ is splitted into multiple smaller mapping problems represented by partly empty comparison matrices. Not all input comparisons within $CM$ need to be included in one of the comparison matrices $CM_1, ..., CM_n$. In the example, a sequence of a Match and a Select operator is executed for each block.

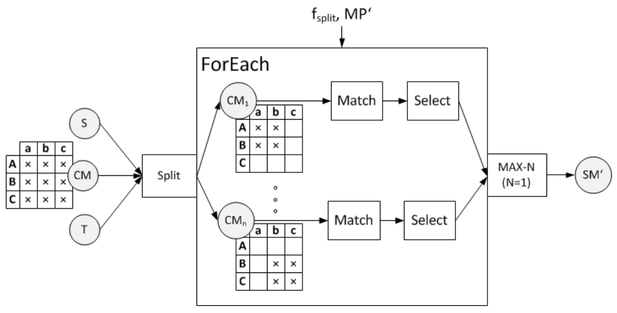


Figure 4.12: For-Each construct: Notation and example

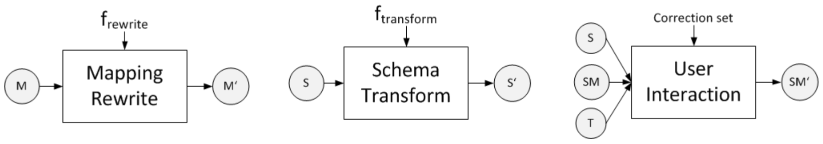


Figure 4.13: Additional operators: Notation and example

### 4.2.4 Further Operators

The operators above form a basis for constructing most internal matching processes of existing systems and offer means to easily design and test new kinds of processes. However, additional operators could be defined to be able to represent special behavior. The basic set of operators could be completed by an Invert and Difference operator that would help to return all those pairs from a similarity matrix that were not matched (have no similarity value greater 0).

Most matching systems perform a number or preprocessing steps after importing the input schemas and post-processing before writing the output mappings. For instance, matching systems preprocess the schema while importing them to some internal data structure. Strings are stemmed and frequent terms are removed to increase string matching quality. In order to represent such behavior a Schema-Transform operator could be included. The operator can be parameterized with a transformation function $f_{transform}$ that can change attributes of a schema but could also remove and add elements and structural relationships to normalize structures. For instance, some relational schemas implicitly include structure into the names of elements (i.e. “AddressName”, “AddressStreet”). Such schemas can be transformed to include an “Address” element as a new parent for “Name” and “Street”. Making the structure explicit could increase the matching quality of structural matchers.

After executing the Create-Mapping operator, post-processing operators could be applied. Mappings could be corrected and changed by a Mapping-Rewrite operator as often done in ontology matching systems. However, in this thesis the primary focus was set onto the match processing so that pre- and postprocessing operators are not further discussed.

An increasingly important aspect of a matching system is the user interaction. As was described in Section 3.3, some systems allow a user to investigate intermediate results and let him correct partial mappings while the matching process is executed. User-corrected mappings can be used to parametrize matching algorithms or do serve as training data for learning. A User-Interaction operator could help here. This operator defines when a user is consulted and what correspondences are presented for correction. The input to the operator is a similarity matrix and the output is a similarity matrix with corrected entries.

## 4.3 Features and the Condition Operator

The major improvement over existing process models for schema matching is the Condition operator that introduces adaptivity. Adaptivity implies that the matching system is able to analyze the input schemas and intermediate results to adapt the execution of the matching process. This is crucial since in practice, schema matching systems are applied onto fully unknown mapping problems. Default configurations, i.e. a predetermined selection of matchers and combination of their match results, are often not robust enough to cope with largely differing mapping problems of diverse domains. In the state of the art literature on schema and ontology matching there are already some attempts to achieve such adaptivity as was discussed in Section 3.2. For instance, the ontology alignment systems RiMOM [99] and Falon [77] compute properties of the input schemas to later select or unselect a structural and a name-based matcher. However, these adaptations are fixed in the code and only deal with a small part of the tuning problem. For generalizing such adaptive behavior of a matching process, the Condition construct was introduced above. It relies on so-called features that are measured from the schemas as well as from intermediate similarity matrices. In the following, different types of features are described in detail.

With features, properties of schemas and intermediate results of a matching process can be computed. In general, features formalize the results of a manual analysis step that a matching expert would do before or while constructing a matching process, e.g. to select and add matchers. A feature is a function that takes one or several schemas and/or similarity matrices as input and computes a positive real value. Most features compute values between 0 and 1 as output. Two general types of features can be distinguished which are schema and matrix features. Schema features try to describe properties of a schema and can be computed in a preprocessing step before actually executing a matching process.

**Definition 11.** ***(Schema Feature)*** *A schema feature is a function $f : S \to \mathbb{R}^+$ that takes a schema $S$ as input and computes a positive real value as output.*

In simple cases, schema features reflect the schema size or the relative frequency of schema element attribute values such as the availability of element annotations or type information. More complex features rely on value distributions of schema elements or structural properties. For instance, the average length of paths in a schema tree gives some indication on when to use a Path matcher evaluating the name similarity of elements and their predecessors.

Some schema features evaluate the degree of similarity between multiple input schemas. These are called Schema Similarity Features.

**Definition 12.** ***(Schema Similarity Feature)*** *A schema similarity feature is a function $f: S_1, \ldots, S_N \rightarrow \mathbb{R}^+$ that takes multiple (N) schemas as input and computes a positive real value as output.*

These features characterize commonalities and differences of schemas. They help to select appropriate matching operators for a specific problem. For example, the structural and linguistic schema similarity feature from RiMOM can be used to decide about the appropriateness of applying a structure-based or name-based matcher.

Matrix features represent properties of an intermediate similarity matrix that is the output of matching operators in the matching process. Matrix features are defined as follows:

**Definition 13.** ***(Matrix Feature)*** *A matrix feature is a function $f: SM, S, T \rightarrow \mathbb{R}^+$ that takes a similarity matrix and the source and target schema as input and computes a positive real value as output.*

They are used to evaluate properties of a similarity matrix. With a reference mapping at hand, the actual quality of an intermediate result could be assessed. However, also without a reference mapping, similarity value distributions or averages of similarity values in a matrix can help to project the quality of a matrix. For instance, a so-called Noise-Feature computes the number of low valued entries in a similarity matrix in relation to the top values in each row and column. The resulting feature-value can be used to assess the quality of a matrix and thus the operator that has generated it. Again, there are matrix features that have more than two matrices as input.

**Definition 14.** ***(Matrix Similarity Feature)*** *A matrix similarity feature is a function $f: SM_1, \ldots, SM_N, S, T \rightarrow \mathbb{R}^+$ that takes multiple (N) similarity matrices and the source and target schema as input and computes a positive real value as output.*

Features with multiple matrices as input often describe the degree of commonalities and differences between these matrices. For instance, a feature could measure the overlap of top-1 values of different similarity matrices. If the overlap is high, more confidence could be put in the different matrices. Matrix similarity features are particularly useful for selecting appropriate combination strategies.

A number of features were already introduced in literature for improving the adaptivity of decision trees [49], for profiling mapping tasks while automatic configuration selection [30], for ontology evaluation [162], for finding combination weights [106] or within Falcon [77] and RiMOM [99] for selecting or unselecting matchers. Most of the proposed features are schema and schema similarity features. Some of them

| Type | Strategy | Description |
|---|---|---|
| S | Attribute-Existence | Specifies the percentage of elements, that carry a {name, type, annotation, instance, cardinality} similar to metrics proposed in [162] |
| | Attribute-Variance | Measures how strong {name, type, annotation, instance, cardinality}- values of different schema elements differ, computed by the entropy of values. |
| | String-Meaningfulness | Meaningfulness of {name, annotation}-values in a schema is measured from querying a search engine such as Google similar to the ideas of probing Wordnet within RiMOM [99]. |
| | Element-Token-Ratio | Measures the number of {name, annotation}-tokens per schema element in relation to the overall number of schema elements |
| | Repeating-Elements | Measures how strong element names and their attribute values are repeated within a schema. |
| | Repeating-Fragments | Measures how strong small schema fragments are being reused within a schema |
| | Schema-Depth | Measures the average path length of all leaves in a schema relative to some maximal length value. |
| | Path-Variance | Specifies the ratio of all distinct paths to the number of all leaves |

Table 4.1: Schema features used or introduced in this thesis

are being reused within this thesis. This is different for matrix and matrix similarity features. Until now, there were not many of such features proposed except for the harmony value for finding weights in a combination [106] or the DICE and SIM-AVERAGE values that were used to assess schema similarity in COMA [11].

### 4.3.1 Schema Features

In Table 4.1 all schema features are listed, that are used within this thesis. Some of them are very simple features like Attribute-Existence that checks availability of certain attributes. But, also more advanced features are contained like Structural-Similarity or String-Token-Overlap. An important property of a feature is that the computational complexity should be low to minimize the impact on the overall execution time. In the table, each feature is described briefly. Only selected features are introduced in more detail below. A complete definition of all listed features can be found in the Appendix A.

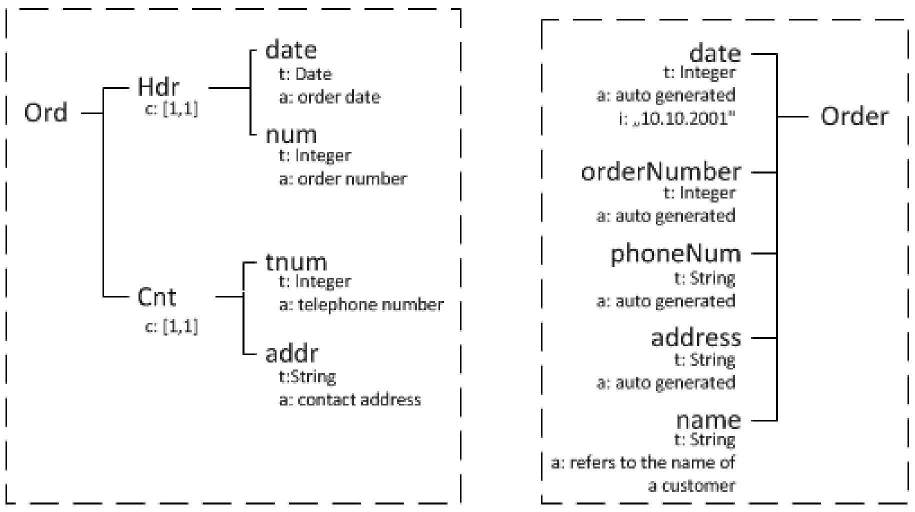


Figure 4.14: Two schema examples

### Attribute-Existence Feature

The *Attribute-Existence* features are simple to compute but are very important for selecting or unselecting matchers. They specify the percentage of elements that carry an attribute $x$ such as name, type, annotation, instance, restriction or cardinality within a schema S:

$$AttributeExistence(S,x) = \frac{\{s \in S \mid s.x \neq \varnothing\}}{|S|} with \;\; x \in \{t,i,a,n,c\} \tag{4.1}$$

Similar metrics where proposed in ontology evaluation by Tartir et. al. [162] with so-called Richness Metrics. They measure the percentage of Attributes, or Object-Properties to Sub-Class relations (Relationship-Richness) or the number of classes for which instances exist (Class-Richness). Tartir et al. also introduce a Null-Label and Null-Comment metric which measures the number of terms that have no label or no comment in relation to the number of terms used in an ontology. Example: In Figure 4.14 two examplary schemas are shown that will be used to give example feature values. The Attribute-Existence for the first schema and its attributes annoation and instances computes as $AttributeExistence(Ord,a) = \frac{4}{6} = 0.66$ and $AttributeExistence(Ord,i) = \frac{0}{6} = 0$. The same feature values for the second schema compute as $AttributeExistence(Order,a) = \frac{5}{5} = 1$ and $AttributeExistence(Order,i) = \frac{1}{5} = 0.2$. Obviously, this basic feature does give some valuable insights into the properties of the given schemas. One could think that an Annotation matcher would be a better choice than an instance-based matcher. However, the Attribute-Existence does not say anything about the quality of existing attribute values.

### Attribute-Variance Feature

The *Attribute-Variance* feature [48] tries to assess the quality of existing attribute values. It measures how well attributes help to disambiguate schema elements while matching. The more attribute values differ, the more information they carry in an information theoretic sense. In order to compute the amount of information contained in a set of attribute values the entropy is commonly used. The entropy originally measures the information contained in a message. It relies on the probability that an item from a set of items is chosen. When adopting this idea for analyzing schemas and attributes the following can be defined: $p(S, x, z)$ refers to the probability that elements in $S$ carry a specific attribute value $z$ for an attribute $x$. With $s.x$ the value of the attribute $x$ of a schema element $s$ is referred to.

$$p(S, x, z) = \frac{\{s \in S \mid z = s.x\}}{|S|} \tag{4.2}$$

The entropy then computes the sum of the product of probabilities $p$ multiplied with the log of $p$. $Y(S, x)$ refers to the set $\{s_1.x, ..., s_{|S|}.x\}$ of possible values of an attribute $x$ in a schema $S$. We then define Attribute-Variance as follows:

$$AttributeVariance(S, x) = \frac{-\sum_{y \in Y(S,x)} p(S, x, y) ld\, p(S, x, y)}{-ld\, \frac{1}{|S|}} \tag{4.3}$$
$$with \quad x \in \{t, i, a, n, c\}$$

The entropy value needs to be normalized by the maximum possible entropy for the schema S that assumes equal probabilities for every $s.x$.
Example: The probability p for the $Order$ schema that an element has the annotation text "auto-generated" is $p(Order, a, "auto\ generated") = \frac{4}{5}$. The probability of carrying "refers to.." is $p(Order, a, "refers\ to..") = \frac{1}{5}$. This leads to the following computation of the Annotation-Variance. $((\frac{4}{5} \cdot ld\frac{4}{5}) + (\frac{1}{5} \cdot ld\frac{1}{5}))/ld\frac{1}{5} = 0.31$. The Annotation-Variance for the $Ord$ schema computes as: $((\frac{2}{6} \cdot ld\frac{2}{6}) + (\frac{1}{6} \cdot ld\frac{1}{6}) + (\frac{1}{6} \cdot ld\frac{1}{6}) + (\frac{1}{6} \cdot ld\frac{1}{6}) + (\frac{1}{6} \cdot ld\frac{1}{6}))/ld\frac{1}{6} = 0.87$. Obviously the Annotation-Variance for the $Order$ schema is low so that an Annotation matcher might have problems to compute reasonable similarity values.

### Other Features

Other schema features that are introduced in this thesis are the *String-Meaningfulness* that is based on Google searches (see Section A.1.1), the *Element-Token-Ratio* that can be used to select or unselect token-based Name matchers with token-weighting [138] (see Section A.1.2), *Repeating-Elements* (Section A.1.3) and *Repeating-Fragments* for identifying reuse within schemas which may lead to n:m mappings (Section A.1.4) as well as *Schema-Depth* and *Path-Variance* (Section A.1.5 and A.1.6) that indicate the relevance of a Path matcher.

| Type | Strategy | Description |
|---|---|---|
| S* | Feature-Similarity | Computes the similarity of two schema features as proposed by Cruz et al. [30]. |
| | Feature-Average | Computes the average of two schema features. |
| | Name-Similarity | Specifies the ratio of similar/equal {name, annotation}-values to non-equal ones within two schemas similar to the linguistic comparability [77], label similarity factor [99] and lexical affinity coefficient [139]. |
| | String-Token-Overlap | String-Token-Overlap tries to determine how similar the set of {name, annotation}-values from the two input schemas S and T are. |
| | Similar-Language | Computes the similarity of the used schema language based on language profiles. (see Section A.2.2) |
| | Structural-Similarity | Computes the structural similarity of two schemas by counting elements with similar structural statistics as proposed in RiMOM [99], Falcon [77] and UFOMe [139]. (see Section A.2.3) |

Table 4.2: Schema similarity features used or introduced in this thesis

### 4.3.2 Schema Similarity Features

Schema similarity features can be built from combining existing schema features. For that purpose, Cruz et al. introduced the *Feature-Similarity (FS)* and the *Feature-Average (AVG)* [30]. *Feature-Similarity* has higher value if the two input feature values are similar whereas the *Feature-Average* simply computes an average. The Feature-Similarity can also be found in the Appendix Section A.2.1. To illustrate the usefulness of the Feature-Similarity the example from Figure 4.14 is picked up again. The average of the computed Annotation-Variance for the $Ord$ and the $Order$ schema could be misleading in having a relatively high value: $\frac{0.87+0.31}{2} = 0.59$. In contrast to that, the Feature-Similarity would compute a value of $0.24$ which tells that the two feature values are quite different. Therefore, choosing an Annotation matcher might not be recommended.

In addition to the two combination features, there are also schema similarity features in the literature that directly compute a single feature value from two input schemas. Some of them are also newly defined in this thesis. Table 4.2 lists all schema similarity features that were used in this thesis. Some of them are described in detail whereas others can be found in the Appendix A.2.

#### Name-Similarity and String-Token-Overlap Feature

The *Name-Similarity* and the *String-Token-Overlap* feature can be used for choosing the most appropriate Name matcher. *Name-Similarity* computes a ratio of similar/equal {name, annotation}-values to non-equal ones within two schemas. It was already proposed as Linguistic Comparability within [77], the Label Similarity Factor within Rimom [99] and the Lexical Affinity Coefficient in [139]. Since the number of similar names is often low in real world mapping problems a token-based feature is introduced. *String-Token-Overlap* tries to project the future quality of a token- or term-based matcher by computing the overlap of token-sets of the two schemas that are to be matched. It determines how similar the set of names or annotations from the two input schemas S and T are:

$$StringTokenOverlap = \frac{|ts(S) \cap ts(T)|}{|ts(S) \cup ts(T)|} \tag{4.4}$$

with $ts(S)$ being the set of all tokens from all element names or annotations of a schema $S$. High overlap values indicate that a {name, annotation}-based matcher could provide good matching results.

Example: The String-Token-Overlap for the element names of the $Ord$ and $Order$ schema computes as follows: $\frac{\{num,name\}}{|\{phone,num,tnum,hdr,cnt,order,address,name,addr,number,date\}|} = 0.18$ which is quite low. The Name-Similarity computes to $\frac{1}{max(6,5)} = 0.16$ since only one element name matches directly.

### Similar-Language

*Similar-Language* tries to compute how similar the used language of two schemas is. This could help deciding to use background knowledge such as Wordnet or dictionaries. Moreover, string matching approaches do only make sense if the matched strings are from a similar language. Names and annotations from different languages possess a profile of character probabilities and word structures which can be exploited to derive the language with high accuracy as done in [47, 111]. By reusing such language profiles, a new feature for schema matching can be defined. More details can be found in the Appendix Section A.2.2.

### Structural-Similarity

*Structural-Similarity* is commonly used to assess how similar the structure of two schemas is. All proposed instances of Structural-Similarity from RiMOM [99], Falcon [77] and UFOMe [139] rely on collecting elements with similar structural statistics. They all ignore name or annotation attributes. More details can be found in the Appendix Section A.2.3.

## 4.3.3 Matrix Features

All matrix and matrix similarity features that are collected in Table 4.3 are newly proposed in this thesis. They can be used to analyze intermediate similarity matrices within the Condition operator to adapt the execution of the matching process while run-time.

| Type | Strategy | Description |
|---|---|---|
| SM* | Selectivity | Evaluates the confidence of a result matrix. It relies on the distance of the top-1 entry in a row or column to the next highest entry in the same row and column. |
| | CrossMatches | Computes how structurally consistent a computed mapping is, i.e. how structurally close the matching target elements of structurally close source elements are. |
| | Node-Position | Computes how structurally consistent a computed similarity matrix is based on comparing the path-length of corresponding elements. |
| | Multi-Matches | Specifies the ratio of n:m matches to the number of 1:1 matches within a similarity matrix. |
| | Monogamy | Measures how close the result is to a 1:1 mapping where element pairs are in monogamic relationships. |
| | Noise | Measures the amount of noise in a matrix based on analyzing a histogram of similarity values. |
| | Sibling-Distribution | Measures the amount of correspondence pairs that link sibling elements in the source schema and also link to sibling elements in the target schema. |
| | Match-Count-Ratio | Measures the amount of correspondence pairs compared to the size of the smaller schema. in the target schema. |

Table 4.3: Matrix features introduced within this thesis

### Monogamy

*Monogamy* was inspired by the Harmony feature. Harmony has problems to cope with matchers that may return many multi matches. Also the actual similarity values are not included in the computation. Monogamy changes this and is able to measure how close the result is to a one-to-one mapping. It assesses how “monogamic” each element pair of a similarity matrix is. A pair is monogamic if each partner of a pair is not involved with other partners of other similarity pairs. A stable marriage is monogamic since every element only has one partner. For each pair of elements in the similarity matrix the number of other pairings is counted where each of the two partners is involved in. The resulting partner matrix P is computed as follows:

$$P(SM) = (p_{i,j})\, with\, p_{i,j} = \begin{cases} \frac{sim(i,j)}{|R(SM,i)|+|C(SM,j)|-1} & sim(i,j) > 0 \\ 0 & sim(i,j) = 0 \end{cases} \tag{4.5}$$

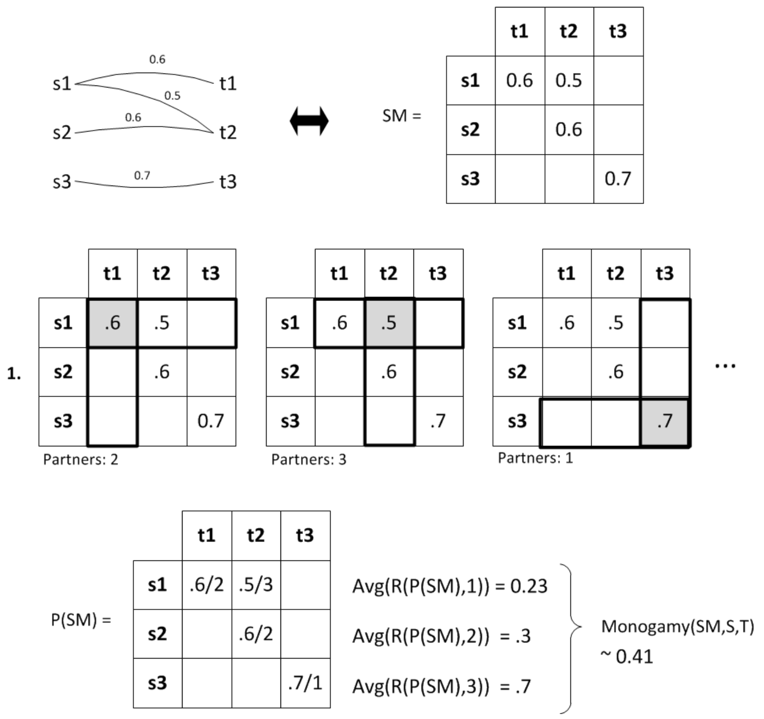


Figure 4.15: Monogamy example with almost perfect mapping

$R(SM, x)$ collects all elements of a row $x$ that have similarity values greater than 0. $C(SM, x)$ does the same for a column $x$.

$$R(SM, i) = \{(i, j) \in SM \,|\, j \in \{1, \ldots, |T|\} \wedge sim\,(i, j) > 0\}| \tag{4.6}$$

$$C(SM, j) = \{(i, j) \in SM \,|\, i \in \{1, \ldots, |S|\} \wedge sim\,(i, j) > 0\} \tag{4.7}$$

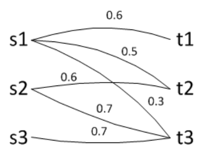
Using these functions and a function Avg(Y) that computes the average of all non-zero values of a set of element pairs Y, the Monogamy feature can be computed:

$$Avg(Y) = \frac{\sum_{(a,b)\in|Y|} sim(a, b)}{|\{(a, b) \in Y \,|\, sim\,(i, j) > 0\}|} \tag{4.8}$$

$$Monogamy(SM, S, T) = \frac{\sum_{i\in\{1,..,|S|\}} Avg(R(P(SM), i))}{|S|} \tag{4.9}$$

Depending on the size of input schemas either the column or the row can be averaged.

Example: Figure 4.15 illustrates how the Monogamy feature is computed. At the top the initial matrix and the corresponding mapping is visualized. To compute the Monogamy all entries of the matrix are iterated and the number of partners (entries with similarity $>$0) in the row and column are counted (1). The partner matrix



⟺

| | t1 | t2 | t3 |
|---|---|---|---|
| **s1** | 0.6 | 0.5 | 0.3 |
| **s2** | | 0.6 | 0.7 |
| **s3** | | | 0.7 |

P(SM) =

| | t1 | t2 | t3 |
|---|---|---|---|
| **s1** | .6/3 | .5/4 | .3/5 |
| **s2** | | .6/3 | .7/4 |
| **s3** | | | .7/3 |

Avg(R(P(SM),1)) = 0.12
Avg(R(P(SM),2)) = 0.18
Avg(R(P(SM),3)) = 0.23

} Monogamy(SM,S,T) ~ 0.17

Figure 4.16: Monogamy example with lower quality mapping

P(SM) is constructed by dividing each similarity value by its partner count. The weighted similarities are then summed up and averaged by row. Finally, these values are averaged to the resulting Monogamy value of 0.41. In the second example from Figure 4.15 additional correspondences are included in the mapping. The Monogamy for that matrix is 0.17 which is lower than before as expected.

### Noise Feature

The *Noise* feature was defined for the following purpose: When analyzing intermediate results it could be observed that many matchers produce low valued similarity values for many element pairs while only few pairs get much higher similarity values assigned. This is particularly problematic if many match results of such kind are combined with a WEIGHTED combination strategy that takes the weighted average of input values. Many small similarity values for one pair could let a pair win against another pair with only few but high similarity values. This could either be solved by introducing a weighting into the combination or by filtering such noise beforehand. Noise is measured with the help of a histogram $h(SM)$ of input values that consists of $B$ histogram buckets $\{h(SM)_1, \ldots, h(SM)_B\}$. The b-th bucket spans an interval and is filled with element pairs as follows:

$$h(SM)_b = \{(i,j) \in SM \,|\, sim(i,j) \in [(b-1)\frac{1}{B}, b \cdot \frac{1}{B}]\} \tag{4.10}$$

From the distribution of values within that histogram the Noise feature value is computed. All bucket sizes are summed up starting with the bucket $b = 1$ until 50% of all pairs are collected. The index of the last summed bucket is kept as $low$.

Then, the number of needed buckets is computed to summarize the highest 10% of similarity values which results in the bucket index $high$.

$$low = b \in B \mid \sum_{x}^{b} |h(SM)_x| > 0.5 \cdot |S \times T| \wedge \sum_{x}^{b-1} |h(SM)_x| < 0.5 \cdot |S \times T| \quad (4.11)$$

$$high = b \in B \mid \sum_{x}^{b} |h(SM)_x| > 0.9 \cdot |S \times T| \wedge \sum_{x}^{b-1} |h(SM)_x| < 0.9 \cdot |S \times T| \quad (4.12)$$

The higher the distance between the $low$ and the $high$ bucket index is the more probable it is that the matrix contains noise. Therefore, the Noise feature value is finally computed as:

$$Noise(SM, S, T) = \frac{high - low}{high} \quad if\, high > low \quad (4.13)$$

In the best case only two buckets are filled which is the first and the $|B|$-th bucket. In that case the distance between the two buckets is high so that also the Noise-value is high. In the worst case only one bucket is filled. In that case high and low bucket indices are similar and the Noise feature value would be 0.

### Other Matrix Features

The remaining Matrix features *Multi-Matches*, *Cross-Matches*, *Node-Position*, *Sibling-Distribution* and *Match-Count-Ratio*, are only described briefly. A detailed definition can be found in the Appendix A.3. *Multi-Matches* is introduced to measure ambiguity within a similarity matrix. This ambiguity could result from reused tokens and elements within a schema which often leads to multi-mappings. In scenarios where mainly 1:1 matches should be contained in the final result the precision would drop. The Multi-Matches feature computes the ratio of n:m matches to the number of 1:1 matches within a similarity matrix. The feature is helpful for implementing the refinement strategy from COMA++. If there are many multi-matches in a mapping then a Path matcher could be used to reduce the number of multi-matches.

*Selectivity* tries to evaluate the confidence of a result matrix that was computed by a matcher or subprocess. It computes the distance of the top-1 entry in a row or column to the next highest entry in the same row and column. The rationale is that a high distance of the best candidate match to the next possible matches implies that the candidate match is certain. A low distance on the other hand shows more uncertainty.

*Sibling Distribution* can be used to decide when to use a Sibling matcher. It also gives a good indication about structural similarities. With respect to the siblings of elements a so-called distribution of matches can be observed. If two correspondences link sibling elements in the source schema and also link to sibling elements in the

| Type | Strategy | Description |
|---|---|---|
| SM* | Commonality | Rate how common a set of input matrices is based on computing and comparing most trustful correspondences from each input matrix. |
| | Complementarity | Measures how strong two matrices complement each other by computing the ratio of conflicting and non-conflicting pairs. |

Table 4.4: Matrix similarity features introduced within this thesis

target schema than a structural similarity of the respective fragments can be assumed. If many such sibling correspondences exist within a mapping then a high structural similarity is present. Based on that observation the Sibling-Distribution feature is defined.

The *Cross-Matches* feature computes how structurally consistent a computed mapping is, i.e. how structurally close the matching target elements of structurally close source elements are. A low structural consistency is an indicator for low precision mappings. In order to increase structural consistency constraint-based selection approaches as proposed in ASMOV [86] could be used. For computing the Cross-Matches feature the ratio of the number of matches that are crossed by some other match to the total number of matches is used.

The *Node-Position* feature also computes how structurally consistent a computed similarity matrix is. Assuming overall structural similarity the path-length of matching element pairs should be similar or differ in the range of a given delta.

The *Match-Count-Ratio* feature computes how many elements of the smaller schema already found a partner. Falcon used a similar feature to assess whether to use additional structural matchers based on the assumption that a 1:1 mapping needs to be computed.

### 4.3.4 Matrix Similarity Features

Table 4.4 lists all matrix similarity features. Currently, only two of such features are provided. For combining similarity matrices as done by the Combine operator a number of combination strategies were proposed. In order to select an appropriate combination strategy the input similarity matrices can be analyzed according to their commonalities and complementarity.

#### Commonality

The *Commonality* feature is able to rate how common a set of matrices is. In order to measure the commonalities the most trustful mapping pairs from each input matrix are computed. Trustful pairs with high precision can be acquired by applying a THRESHOLD selection with high threshold value followed by a MAX-N and EXACT

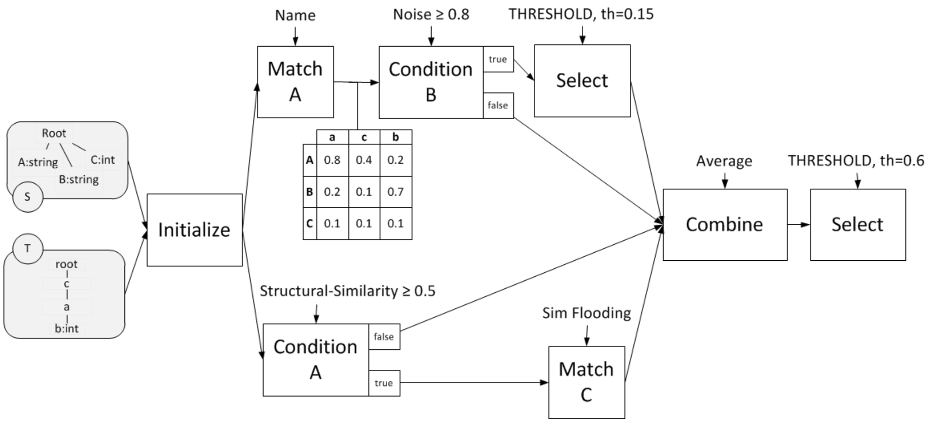


Figure 4.17: Example matching process with conditions

selection. For trustful pairs the agreement between all input matrices is computed. High values of the Commonality feature imply that all matrices agree on many pairs whereas no agreement would lead to a small value.

**Complementarity**

*Complementarity* is introduced since a low Commonality feature value does not directly suggest that the matrices should not be combined. It happens that different matchers compute similarities for different parts of a schema. If those mapping fragments are to be combined, their commonality is low, but they could be complementary. This means that pairs could complement each other instead of conflict. Pairs are conflicting if they point to the same source element but to different target elements or if they point the same target elements but different source elements. From that assumption the Complementarity feature is derived. It also relies on selecting the trustful pairs from every input matrix. The ratio of conflicting pairs to all other pairs then computes the Complementarity value. If input matrices are not common and also do not complement each other, then some input matrices should not be included in the subsequent combination.

## 4.3.5 Example

By using the matching process operators including the Condition construct, a simplified version of the internal process RiMOM can be modeled that selects a structural matcher based on the Structural-Similarity feature. (see Figure 4.17). To illustrate the use of matrix features, a condition is added that relies on the Noise feature. Such conditions are newly introduces in this thesis and were not part of internal processes of existing matching systems.

The matching process relies on two matchers which is a Name matcher and Similarity-Flooding as structural matcher. Initially, schemas are imported and an empty similarity matrix is initialized. Then the Match operator A and Condition A are executed. Condition A computes the Structural-Similarity feature value. Due to the low structural similarity of the two input schemas the Condition evaluates to false. The subsequent branch with the Match operation C is ignored. The result of the Match operator A contains many low similarity values. The subsequent Condition computes the Noise feature and evaluates to true, since the Noise is bigger than the given 0.8. The branch following on the false output is set to ignore, the branch that follows the true output is left unchanged. The subsequent selection is executed with a low valued threshold of 0.15. Finally, the Combine operator combines the results of the three different branches. Two of these branches are ignored. The selection extracts pairs with similarity above a given threshold. Afterwards, a mapping could be created by using the Create-Mapping and Export-Mapping operators.

## 4.4 Matching Process Design Patterns

The presented model enables a user to build tailored problem-specific matching processes. This involves choosing appropriate operators, their ordering and parameterizations which is mostly based on experience. However, such experience is often hidden in the internal processes of existing systems. From analyzing these systems a collection of so-called process design patterns can be identified as was described by the author [133]. The idea of a matching process design pattern is similar to design patterns in software engineering. They should help a designer to develop strong matching systems with small effort that are adapted to a given use case or domain.

In this section, some of the most relevant patterns from [133] are described shortly. Also the results of comparing the behavior of different patterns are briefly summarized.

### 4.4.1 Identified Set of Patterns

The parallel, sequential and iterative topologies that were described in Section 3.1 already form a basic set of design patterns. This set was extended by Peukert [133] with a number of patterns such as Conditional Execution, Skimming, Refinement or Divide and Conquer. The Conditional Execution was already introduced in Section 4.2.3 without calling it a matching process design pattern. The other three patterns Skimming, Refinement as well as Divide and Conquer are described in the following.

*Refinement* is can be used to improve the results of a matcher within subsequent matchers in a process as proposed within COMA [11]. It is applied if matchers produce mappings with typically high recall but low precision. Refinement starts the process with a matcher producing high recall and increases the precision by using subsequent matchers that are based on other schema element attributes. Refinement is constructed with Filter operators (see Figure 4.18). The result similarity matrix $SM_1$ of the first matcher $m_1$ is filtered using some filter strategy $f$ such as a threshold. For all entries of the first similarity matrix that have a value smaller than a given

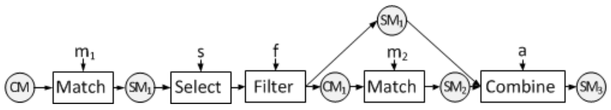


Figure 4.18: Refinement Sequence pattern

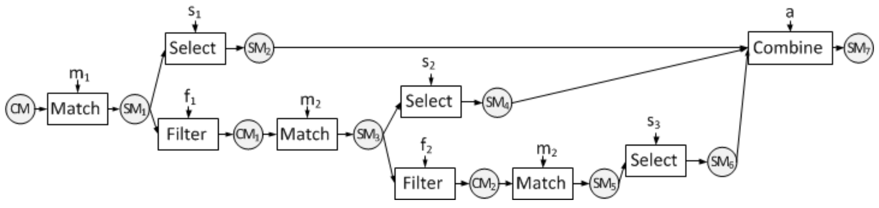


Figure 4.19: Skimming Proccess pattern example

threshold the comparison matrix entry is set to false. The new comparison matrix $CM_1$ is then put into a second matcher $m_2$ to refine the found correspondences. In practice, the result of the first matcher is often reused within a subsequent Combine operator in order to not throw away computed similarity values. In COMA a separate operator called refine was used to put node-based and structure-based matchers in a sequence. Also, the performance improvement approach that is described in Chapter 6 strongly relies on the refinement pattern. It automatically transforms parallel patterns into refinement patterns, without loss of quality.

The rationale of *Skimming* is to extract the most probable correspondences from every matcher individually. These correspondences are “skimmed”. This approach is useful if individual matchers or selection strategies are known to achieve a high precision for a domain of mapping problems. The Skimming pattern consists of multiple Match, Filter and Select operators. Figure 4.19 shows a skimming pattern with three matchers. The results of individual matchers are selected with high precision selection strategies such as high threshold. The selected result is directly put into the final result combination. This lowers the risk to lose correct correspondences in subsequent operations. All other comparisons are filtered with a Filter operator and left for matching with the next matcher. Filtering means that comparisons for skimmed correspondences can be set to false in the comparison matrix which reduces the search space and helps other subsequent matchers to focus on the “harder” comparisons. Again, the result of that second matcher is selected and put into the final result. Skimming can also be used to prune out comparisons where no correspondence is expected upfront. This also reduces search space and increases precision of the overall process. The intention of skimming has some similarities to the rationale of “Boosting Schema Matching” from Marie and Gal [109]. In their work, machine learning is used to iteratively add matchers to an ensemble based on a computed error. From the error a weight is computed that is later used to combine a matcher in a parallel composition. With each iteration, the weights get lower. The higher the weight of a matcher the more correspondences are “skimmed” from

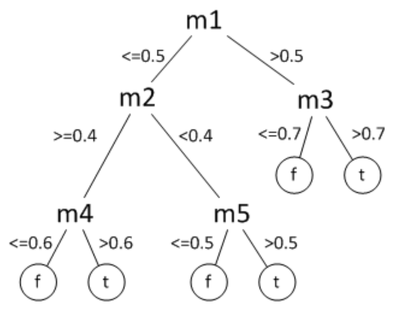


Figure 4.20: Decision tree example

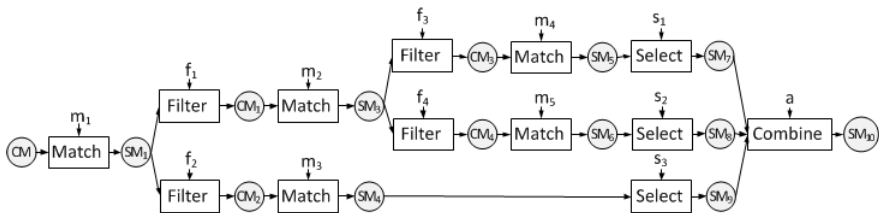


Figure 4.21: Decision tree represented as matching process

that matcher result. Skimming was also applied internally within Falcon where good results of node level matchers are directly added to the final result. If correspondences are still missing, a structural matcher is executed that adds further matches to the final result.

The *Divide and Conquer* pattern is used to divide the set of comparisons based on some condition and distribute these comparisons to the most appropriate, possibly different matchers. For instance, matching ontology properties could be done differently from matching classes or instances. Processes that heavily rely on Divide and Conquer are decision trees. Decision trees can be manually defined, but are often constructed by using machine learning techniques. Figure 4.20 shows a decision tree as it could be learned within MatchPlanner [43]. Such trees can be represented as a matching process by using operators of the matching process model and the comparison matrix from above (see Figure 4.21). First, a matcher $m_1$ is executed. The filter operator sets all entries in the comparison matrix $CM_1$ to false if the filter condition $f_1$ is not met. This comparison matrix is then input to the next matcher $m_2$. In a decision tree the filter condition $f_2$ is typically the negation of $f_1$ ($f_2 = \neg f_1$). Thus, for all other entries the lower branch with matcher $m_3$ is executed. All Match operators recompute the similarity matrix and do not include constituent match results. The final result is constructed by combining all selected correspondences from the leaf nodes of the tree. The bottom-most matchers actually compute the final similarity value.

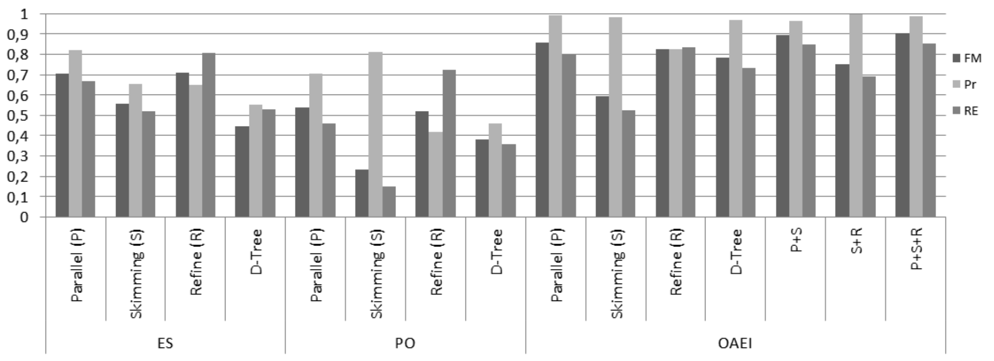


Figure 4.22: Comparing complex patterns and their combinations

### 4.4.2 Evaluating Patterns

In [133] an evaluation of the abovementioned matching process design patterns and their combinations was performed. As data set the mapping problems from Section 2.5 were used. In the experiments seven matchers were taken to construct different matching processes that rely on one of the patterns from above. In Figure 4.22 the results of comparing individual patterns and their combinations with each other on each data set are summarized.

Surprisingly, among the individual patterns, the process that applies the Parallel pattern returns best f-measure results with all data sets. The Skimming-based process is often able to return higher precision at the cost of recall which is characteristic for Skimming. The precision is mostly higher than the recall. In contrast to Skimming the recall in the Refinement-based process is higher, as expected. The decision tree process performed best on mapping problems that are similar to the problems that were used for training.

In a further experiment, the strength of different patterns were combined and compared on the OAEI Benchmark. A combination of Parallel and Skimming (P+S), Skimming with Refinement (S+R) and a combination of all three patterns (P+S+R) was created. In Figure 4.22 the result of that evaluation is shown. As expected, the precision of the Parallel process can be improved by skimming correspondences with very high similarities beforehand (P+S). Also the recall improves. Combining all three patterns further improves the recall and precision. However, a strong process should always include parallel components since a combination of Refinement and Skimming (S+R) is still inferior to Parallel (P).

In a final experiment the similarity values of found correspondences using the different process patterns on an exemplary mapping problem from the PO set (see Figure 4.23) was investigated. The Parallel process returns correspondences with low but similar similarity values (a). This low level of similarity can be explained by an Instance and an Annotation matcher that do not return values on the PO dataset and therefore lower the average that is computed by the Combine operator.

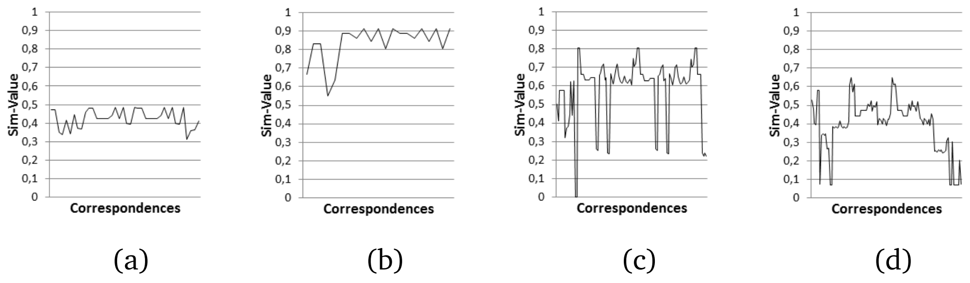


Figure 4.23: Comparing result similarity values of matches computed with different patterns

The Skimming (b) only takes correspondences with high value from each matcher. Therefore the final results are mostly between 0.8 and 1. However, the overall number of correspondences is typically lower which reduces the recall. The decision tree (c) shows a characteristic behavior. Only the similarities of the bottom-most matcher in the decision tree are taken for the final result. The effect is that some correspondences have values that are close to 0. The interpretation of these similarity values for later use is problematic. Obviously, the result does not follow statistical monotonicity [107] which restricts reuse of match results for further operations. Statistical monotonicity measures how well the similarity values of correspondences correlate with the probability of being a correct result.

Refinement also returns rather low similarities, since again the last matcher defines the similarity, which not necessarily gives good estimates.

The presented set of patterns already shows the strength of the matching process model to build very different processes with different properties and quality goals. The evaluation shows that there is some potential to improve the matching quality by solely changing the structure of a matching process.

## 4.5 Matching Process Execution Framework

A Matching Process Execution Framework is introduced that is able to execute matching processes and also tries to foster reuse. The overall architecture is shown in Figure 4.24.

The system gets a source and target schema as input and computes a mapping as output. The core of the framework is an operator library consisting of the operators mentioned above such as Match, Combine or Select. For each operator, specific implementations of Matchers, Selection and Combination strategies can be added as loosely coupled plug-ins. Plug-ins implement a thin adaptation layer that calls components of existing systems and converts inputs and outputs of these components into internal framework formats. In Figure 4.24 some exemplary plug-ins are shown which could be matchers from the Falcon or COMA system, the OWA combination strategy or the MAX-DELTA and MAX-N selection strategy from COMA. Using the

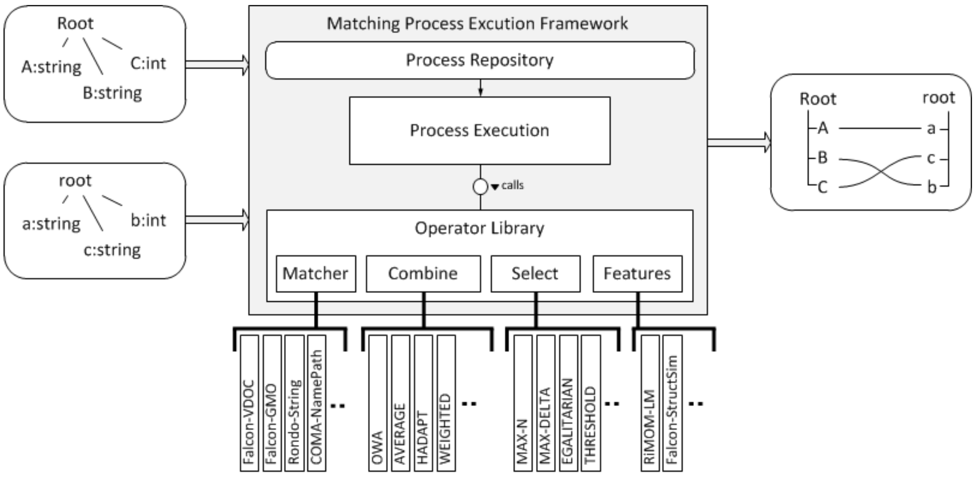


Figure 4.24: Matching Process Execution Framework

above mentioned set of operators together with the registered matchers and strategies, matching processes can be defined that combine components from different matching systems. The created matching processes are executed by the Process Execution that calls the respective operators from the library as is described in the next section.

### 4.5.1 Matching Process Execution

When executing matching processes that contain a Condition operator special attention has to be put on the execution algorithm of the matching process. Instead of changing the data (similarity matrix, comparison matrix, schemas or mappings) that flows through the matching process, the Condition influences the control flow. Some branches in the process are not executed if a condition expression evaluates to false. The Condition construct has an important relation to the Combine operator: A branch after a Condition is ignored until the end of the matching process or if it ended in a Combine operator that has input from a branch that is not ignored. The Combine operator treats the input similarity matrix as not existing and only combines the remaining ones. Thus, the Combine operator always closes the branch that was created by a Condition if there is an input branch to the Combine operator that is not ignored.

The Process Execution component implements the *ExecuteProcess*-algorithm from Algorithm 4.1. The algorithm relies on a stack that is filled with not yet executed operators while processing. It ensures that operators only execute when all incoming operators were already executed which is marked by a state EXECUTED or if they are in IGNORED state.

Initially, an operator, if possible a Schema-Import operator, is pushed to the stack (Line 2) and the execution starts by calling *ExecuteNext* (Line 3). In Line 7-8, the operator that is on top of the stack is analyzed. The function CanExecute() checks

**Algorithm 4.1** ExecuteProcess

```
ExecuteProcess(MatchingProcess m)
  s.push(m.operators[0])
  ExecuteNext(s)
END

ExecuteNext(Stack s, MatchingProcess m)
  c <- stack.pop()
   IF CanExecute(c) THEN
    IF NOT allIncomingIgnored(current) THEN
      c.execute()
      c.state <- EXECUTED
      IF c.type=Condition AND c.result=false THEN
        c.state=IGNORED;
    ELSE
      c.state  <- IGNORED
    stack.push(c.following)
    IF NOT s.empty() THEN}
      ExecuteNext(s, m)
  ELSE
    s.pushToBottom(current)
    s.push(c.predecessor)
    ExecuteNext(s, m)
END

CanExecute(Operation o)
  foreach p IN c.predecessor
    IF NOT (p.state=EXECUTED OR p.state=IGNORED) THEN
    return false;
  return true;
END
```

if all predecessors are either EXECUTED or IGNORED. Line 9 checks that not all incoming operators are ignored. In case all incoming operator are in IGNORED state, the current operator is also set to IGNORED. The IGNORE state is propagated to the current node if all incoming nodes are not ignored. If there is at least one incoming operator that is not in IGNORED state then the operator is executed in Line 10 and its state is set to EXECUTED (Lines 11). Hence, if a combination node has multiple input edges and one of these input edges is not in IGNORED state then the Combination can be executed. This closes the ignore branch that was opened by the condition. If the current operator $c that was executed is a Condition that evaluated to false (Line 7) then the positive output node is set to IGNORE. In Line 16, all following nodes of the current operators are pushed to the bottom of the stack. If the stack is not empty, *ExecuteNext* is called recursively (Lines 17-18). In case the operator could not be executed due to not yet executed incoming operators it is pushed back to the bottom of the stack together with all its predecessors and *ExecuteNext* is called again (Lines 19-22). The process ends when all operators of the matching process were executed or ignored.

## 4.6 Advances over Related Work

The introduced matching process model improves existing process models from Lee et al. [98] and Pirra and Talia [139, 24]. It also goes beyond the set of operators from existing matching process models [183, 104, 178] in a number of dimensions.

First, the proposed set of operators is more complete and allows modeling the behavior of a variety of existing matching systems by using a Filter, Condition, Loop and For-Each. The Condition operator introduces means to model adaptive behavior of a matching process. This allows modeling the internal process of existing matching systems and reoccurring matching process design patterns that are used in Falcon or RiMOM. This was not possible with existing proposals for matching processes. Even decision trees can be represented as a matching process. Still, these trees can get very complex and unreadable for a matching process expert. An important contribution are the set of presented features. In particular the matrix and matrix similarity features are novel. They can be used to let a process adapt to properties of intermediate results.

Second, the introduction of a comparison matrix together with a Filter operator allows to improve performance of a matching process. In Chapter 6 this Filter operator is extensively used to automatically optimize the performance of a matching process.

Finally, the matching process model allows for reuse of existing matching components. When experimenting with the proposed plug-in architecture, a number of existing components (matchers, selection strategies, combination strategies, features) from COMA, Rondo, Falcon and others could be integrated for evaluation purposes into the framework as loosely coupled plugins without touching the original components. It is now possible to build a matching process that relies on components from these different matching systems.

Still, the Condition introduces further complexity in modeling a matching process. The more matchers a matching process contains the more conditions could be created. It is already obvious that it is impossible to create the perfect matching process that contains many conditions to let the process adapt to any input schema type or behavior of matchers and their intermediate results. Furthermore the conditions and their expressions need to be parameterized which is difficult, even for a matching expert. Obviously, some graphical support in modeling matching processes is needed as is described in the following Chapter 5. Moreover, automating parts of the matching process construction would be helpful. Such automation is the major focus of the second part of this thesis.

# Chapter 5

# Graphical Modeling of Schema Matching Processes

The matching process model that was introduced in the previous chapter allows a user to model complex domain specific matching processes. Still, the construction of these processes is mostly done programmatically. This can be cumbersome in particular with large size matching processes that consist of many Conditions, Loops and For-Each constructs. Intermediate results are not easy to investigate since these are typically represented as 2 dimensional arrays of float values. Often this has to be done with a debugger on the programming level.

In this chapter a graphical user interface, the so-called Matching Process Designer, is described. The Matching Process Designer visually supports the construction and tuning of matching processes. Intermediate results can be analyzed by using multiple visualization techniques. A process can be executed in a step-wise way and parameters can be changed on the fly. In addition to the modeling features, the Matching Process Designer consists of a component for comparatively evaluating manually built matching processes. This helps to assess the behavior of processes when solving different mapping problems. Since the Matching Process Execution Framework from above allows integrating matching components from different matching systems, a comparison of the behavior and match quality of different matchers, combination and selection strategies as well as blocking methods can be done visually.

## 5.1 Graphical Modeling of Processes

Matching processes are modeled in a graphical process editor by dragging operators from a tool-box of operators onto a design surface and by connecting them. A screen shot of the toolbox and process editor is shown in Figure 5.1.

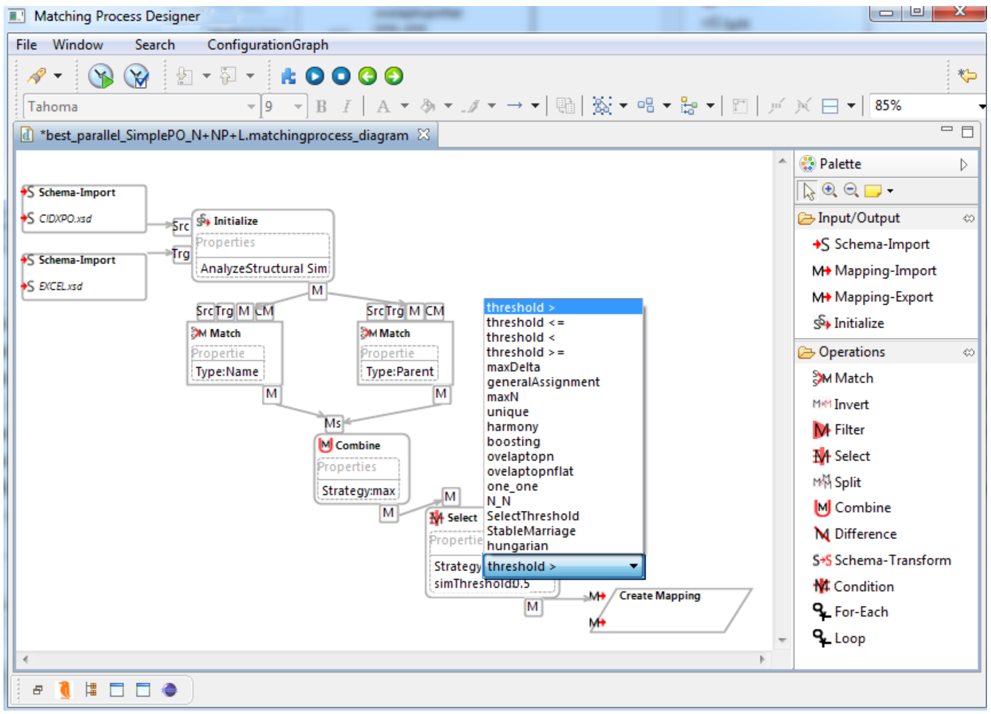
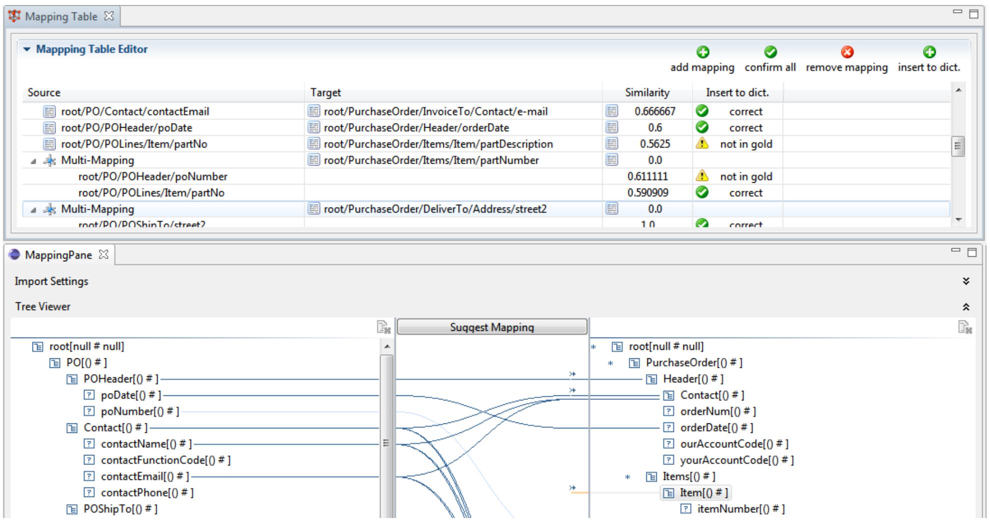


Figure 5.1: Toolbox and graphical process editor

The implementation of the tool was done using the Eclipse Rich Client Platform[1] and the Graphical Modeling Framework (GMF)[2].

Simple graphical representations were defined for the operators of the matching process model. Each operator has specific input and output ports that can be connected to other operators. They are marked as small attached boxes. For instance, the Match operator could get the source and target schema, or an input matching result as input. The output of the match operator is a similarity matrix. The positioning of the boxes allow a user to model a matching process in horizontal as well as vertical direction. Operators are connected through arrows that represent the data-flow in the matching process. An arrow combines a similarity matrix, a comparison matrix and also the source and target schema. Individual parameters of operators can be set directly in the graph as shown for the Select operator in the screen shot. The screen shot also shows a simple and complete matching process. In that process, schemas are imported and an empty matching result is initialized. Two Match operators are executed and the matching result is selected with a THRESHOLD selection strategy. Finally, the result mapping is created.

[1]http://www.eclipse.org/home/categories/rcp.php
[2]http://www.eclipse.org/modeling/gmp/



| Source | Target | Similarity | Insert to dict. |
|---|---|---|---|
| root/PO/Contact/contactEmail | root/PurchaseOrder/InvoiceTo/Contact/e-mail | 0.666667 | ✓ correct |
| root/PO/POHeader/poDate | root/PurchaseOrder/Header/orderDate | 0.6 | ✓ correct |
| root/PO/POLines/Item/partNo | root/PurchaseOrder/Items/Item/partDescription | 0.5625 | ⚠ not in gold |
| Multi-Mapping | root/PurchaseOrder/Items/Item/partNumber | 0.0 | |
| root/PO/POHeader/poNumber | | 0.611111 | ⚠ not in gold |
| root/PO/POLines/Item/partNo | | 0.590909 | ✓ correct |
| Multi-Mapping | root/PurchaseOrder/DeliverTo/Address/street2 | 0.0 | |
| root/PO/POShipTo/street2 | | 1.0 | ✓ correct |

Figure 5.2: Mapping visualization

## 5.2 Tuning of Matching Processes and Analysis of Intermediate Results

In addition to modeling, the built matching process can be executed within the tool to test and tune its parameters. A schema view represents schemas as trees that could have multiple root nodes. Source and target schema fragments can be selected and a chosen matching process can be executed. If a mapping between both schemas is given, the result quality is automatically computed as precision, recall and f-measure.

Moreover, a matching process can be executed in a stepwise way. The user can step forward and backward in the process or set breakpoints. The currently executed operator is highlighted and the intermediate matching result is shown for analysis. The tool provides three mapping visualizations that complement each other. A table-based visualization (see Figure 5.2), which shows source and target elements grouped by source elements together with computed similarity values and their relation to an optionally given correct mapping. Source and target can be exchanged so that also groupings by target elements are possible. The grouping helps a user to investigate multi-mappings.

Secondly the mapping is visualized as connecting lines between elements in the source and target tree views. If elements or lines are selected the respective correspondences in the table-view are selected too. Correspondences that are correct according to a gold standard have darker color. The two aforementioned visualizations help the user to investigate computed similarities of individual schema element pairs. When investigating intermediate (not yet selected) matching results with thousands of correspondences, line-based and table-based visualizations provide too much information for the user. For that reason a new visualization of intermediate results is proposed (see Figure 5.3 for examples). It reuses the Lego-Plot Demo from FreeHEP, which is a 3D visualization framework [3] that allows plotting three dimensional graph data. When applied for visualizing similarity matrices a novel kind of interaction with intermediate mapping results is possible.

It represents an intermediate matching result as a cube that can be moved, zoomed and turned around in 3D with the mouse pointer. The x- and y-axis represent the source and target elements and the z-axes represents the similarity values. The cube gives an aggregated representation of mapping pairs. It turned out to be quite intuitive for analysis of large intermediate results. The user can analyze the overall distribution of similarities of matching results. Also, conclusions about the selectivity of the results can be drawn. A selective result consists of a number of steep peaks whereas a non-selective result produces flat blocks. In many cases the cube representation produces steps that give a good indication of possible thresholds for later selection and filtering. At the bottom of the top-left example cube of Figure 5.3 many low-valued similarities can be found that were classified as noise in Section 4.3.3. A number of similarities are higher than 0.9 and only few more are higher than 0.6. These values can be taken as threshold in the Select and Filter operator. Each matcher produces a characteristic representation in the visualization that tends to be similar for different test cases. Some matchers produce results that show a big variance of similarity values (see top-left cube in Figure 5.3 of the Name matcher result), whereas for other matchers like the Path matcher the variance is much smaller (see top-right cube in Figure 5.3). The cube at the bottom shows a Name matcher result of a larger mapping problem.

The combination of step-wise execution, intermediate result analysis and direct parameter changes allows a user to easily tune a matching process for a given mapping problem. However, in order to build robust matching processes a user needs to evaluate a modeled matching process on sets of mapping problems which is supported by the following evaluation component.

## 5.3 Comparative Evaluation of Matching Processes

The Matching Process Designer offers a component for evaluation. In that component, a number of matching processes can be selected to compare their quality on a selected set of mapping problems. The tool then directly starts an evaluation, computes precision, recall and f-measure and visualizes the results. An exemplary evaluation is shown in Figure 5.5. On the left a number of test mappings from the COMA evaluations and six different matching processes are selected.

On the right, the evaluation result is visualized. Such visualizations are directly generated within the tool. As can be seen, the result quality of the individual processes on the given data set is quite different. A user can now pinpoint shortcomings of the selected matching processes if a gold standard exists. As described in Section 4.5 the

[3] http://java.freehep.org/

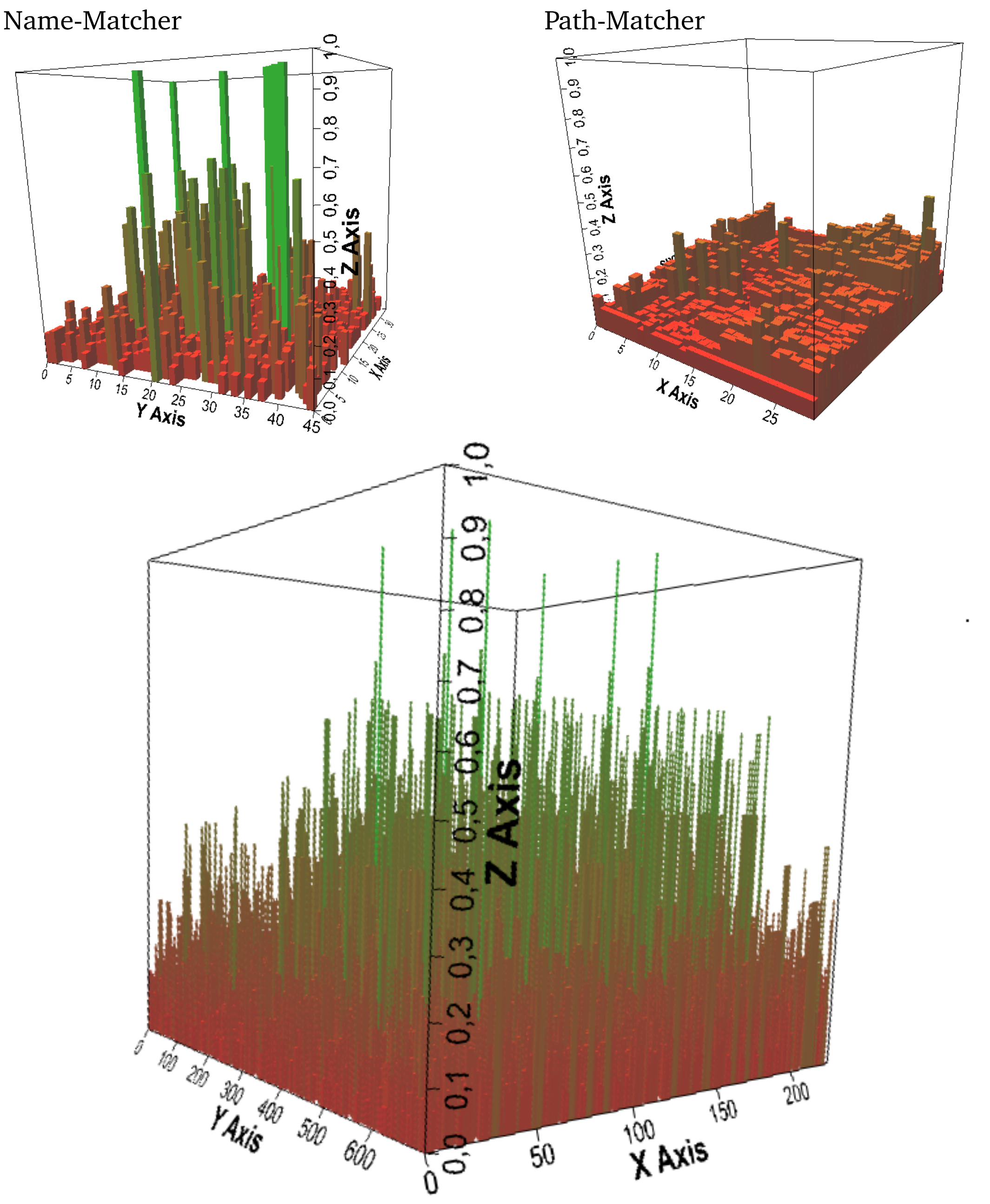
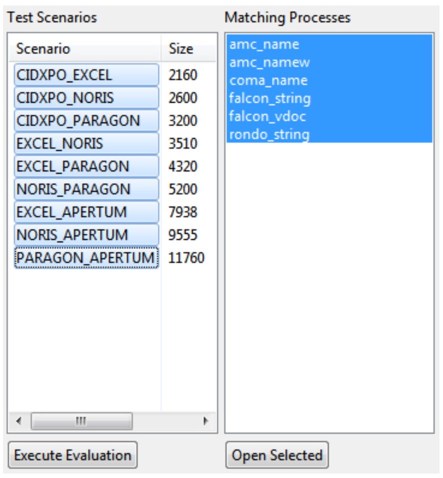


Figure 5.3: Cube of similarities



Test Scenarios

| Scenario | Size |
|---|---|
| CIDXPO_EXCEL | 2160 |
| CIDXPO_NORIS | 2600 |
| CIDXPO_PARAGON | 3200 |
| EXCEL_NORIS | 3510 |
| EXCEL_PARAGON | 4320 |
| NORIS_PARAGON | 5200 |
| EXCEL_APERTUM | 7938 |
| NORIS_APERTUM | 9555 |
| PARAGON_APERTUM | 11760 |

Matching Processes

amc_name
amc_namew
coma_name
falcon_string
falcon_vdoc
rondo_string

Figure 5.4: Configuring evaluation of matching processes

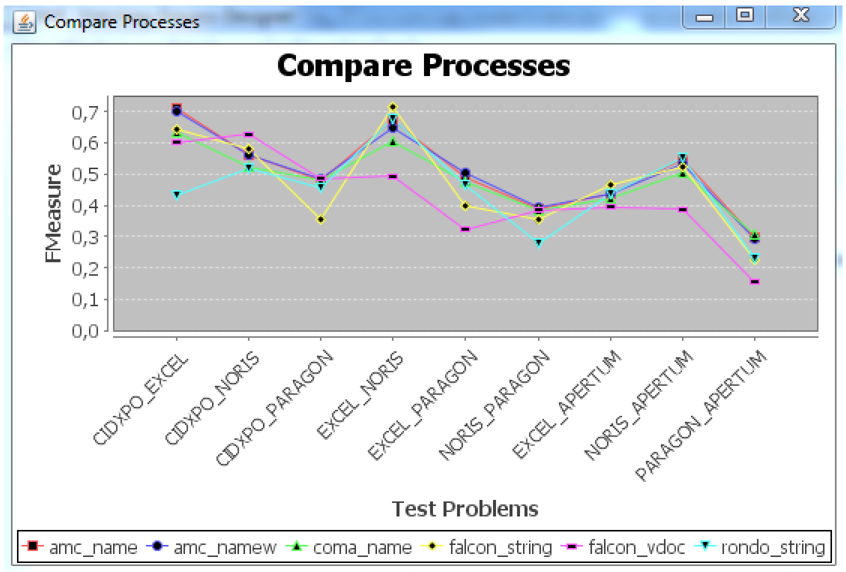


Figure 5.5: Exemplary evaluation result of matching processes

Matching Process Execution Framework is able to integrate individual components of different matching systems. Thus, components originating from different matching systems can be compared. This helps the user to better understand the behavior of these systems at the level of individual matchers, selection or combination strategies. The exemplary evaluation from Figure 5.4 and 5.5 helps to demonstrate that feature. The results are described below.

One observation that could be made is that many matching systems implement their own name-based and structural matchers. Thus, almost no reuse is done across matching systems. The evaluation from Figure 5.5 compared different simple matching processes consisting of a Name matcher and a MAX-N-Selection operator (N=1). The processes only differ in the used name-based matcher and the following implementations were tested: Falcon-String, Falcon-VDOC, AMC-Name [135], AMC-NameWeighted [138], Rondo-String, COMA-Name. As can be seen, the result quality of the individual matchers on the given data set is different across the basic Name matcher implementations. In comparison to the others, the AMC-Name matcher was the favorable matcher for the given test cases. However, the other matchers are closely following. In some cases the Falcon-String matcher had difficulties to compute a good matching result. These are often cases where a token-based matcher is more appropriate. The VDOC matcher from Falcon performed even worse. VDOC collects labels, documentation of an element and its neighbors into a text document and applies a TF-IDF like approach to compute similarities. The matcher has problems with the given task since it only looks for exact word matches which are rare in many XSD-schema mapping problems. The Rondo-String matcher did also have problems with some test-cases. With the help of the process designer the intermediate results of the Rondo-String matcher could be investigated further. Since also other components can be integrated in the Matching Process Framework more advanced evaluations are possible that compare blocking operators such as COMA++ Fragmentation [11] and Falcon partitioning [77] or propagation-based structural matchers like Similarity Flooding [117] or Falcon-GMO [77].

## 5.4 Advances over Related Work

The introduced Matching Process Designer contributes to a research area where not much work has been done before. With the proposed tool, individual matching system development changes from programming on the code level to graphical dragging and dropping operators on a surface. This significantly simplifies and speeds up the construction and tuning of a matching system. The process model is made explicit and the process structure as well as parameters can be graphically manipulated to create a variety of matching processes for different domains. The proposed model and visualization seems to be more intuitive than the existing cell-based visualizations from Bernstein et al. [24]. The UFOMe system [139] allows a user to graphically model matching processes on a similar abstraction level. However, the choice of operators is limited and adaptivity as well as performance aspects cannot be modeled.

Moreover, the system lacks facilities for tuning and debugging which is crucial for improving the quality of a modeled process.

With regards to visualizations of intermediate results only one group published similar work. Cruz et al. [32] very recently introduced a visual analytic panel that shows similarity values of a matcher in a matrix and marks correct and incorrect matches. The cube-based visualization within the Matching Process Designer is superior to existing approaches since it relies on a third dimension within the z-axis to represent similarity values. Moreover, by synchronizing a table- and line-based view the best of two visualization approaches os exploited with the Matching Process Designer. Finally, the evaluation facilities for comparing existing matching processes and components of existing systems are novel. They help to investigate the quality of existing matchers on given mapping problems which could foster reuse when constructing new matching processes. This feature goes beyond the state of the art that typically compares whole systems with default configurations with each other. In such approaches the shortcomings and strength of individual systems cannot be identified. Some systems perform well due to their combination and selection methods whereas others have very strong matchers.

Still, some issues remain. Constructing a matching process still requires expert knowledge about schema matching. Matching processes can get large, in particular if conditions are used to select or unselect parts of a process. What is still missing is automatic optimization of manually designed matching processes. Such mechanisms are investigated within the third part of the thesis. There, a rewrite-based approach is used to automatically improve the performance of a matching process. In a second step complete matching processes are constructed fully automatically.

## Part III

# Rewrite-based Process Tuning and Construction

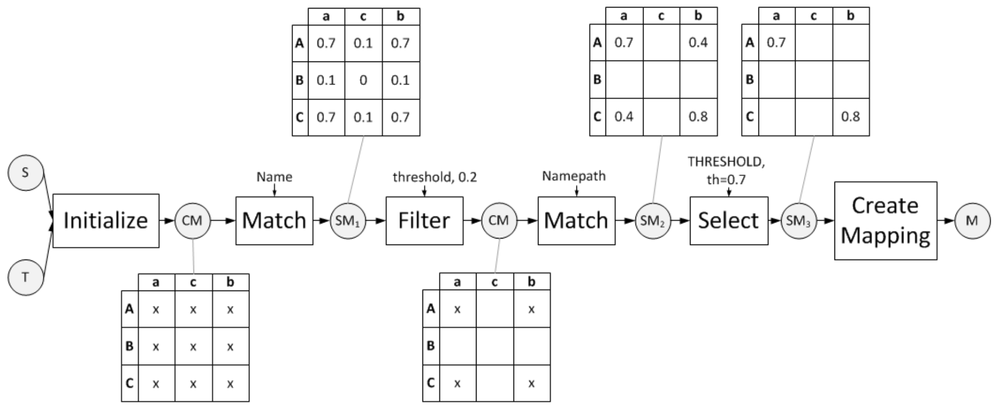
# Chapter 6

# Performance Oriented Matching Process Rewrite

In this Chapter the Filter operator and its filter strategies are investigated more thoroughly. A number of filter strategies are described in Section 6.1. With the help of these strategies sequential combinations of existing matchers are evaluated in Section 6.2. From these evaluations a new generic rewrite-based approach for significantly improving the performance of matching processes is derived. The approach is presented in Section 6.3 together with a simple cost model to compute the relevance of so-called matching rules. The chapter closes with a comparison of the new rewrite-based optimization to existing work.

## 6.1 Comparison Filtering

In Chapter 4 the comparison matrix and the Filter operator were introduced. They can be used to model performance aspects on the process level. Figure 6.1 gives an example of applying the comparison matrix in a sequential matching process. Initially, two schemas are matched using the Name matcher. Within the Filter operator, element pairs with a similarity lower than 0.2 are pruned out by setting the cell in the comparison matrix to false (visualized as a cross in the bottom matrix). The following Namepath matcher is only computing similarities for the remaining comparisons that are still in the comparison matrix. In the example, more than half of the comparisons are pruned out. Note that possible matches might be pruned out early even though they could be part of the overall mapping result. This behavior could drop recall but could also increase precision. An optimal filter would maximize the number of pruned element pairs and at the same time minimize the number of wrongly pruned comparisons that are part of the final mapping. The optimal solution would only drop comparisons that lost the chance to survive a later selection. In the following, different filter strategies are discussed in detail.



| | a | c | b |
|---|---|---|---|
| A | 0.7 | 0.1 | 0.7 |
| B | 0.1 | 0 | 0.1 |
| C | 0.7 | 0.1 | 0.7 |

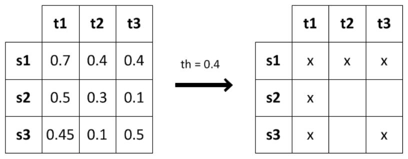
| | a | c | b |
|---|---|---|---|
| A | 0.7 | | 0.4 |
| B | | | |
| C | 0.4 | | 0.8 |

| | a | c | b |
|---|---|---|---|
| A | 0.7 | | |
| B | | | |
| C | | | 0.8 |

| | a | c | b |
|---|---|---|---|
| A | x | x | x |
| B | x | x | x |
| C | x | x | x |

| | a | c | b |
|---|---|---|---|
| A | x | | x |
| B | | | |
| C | x | | x |

Figure 6.1: Example of filter process and its comparison matrix



| | t1 | t2 | t3 |
|---|---|---|---|
| s1 | 0.7 | 0.4 | 0.4 |
| s2 | 0.5 | 0.3 | 0.1 |
| s3 | 0.45 | 0.1 | 0.5 |

| | t1 | t2 | t3 |
|---|---|---|---|
| s1 | x | x | x |
| s2 | x | | |
| s3 | x | | x |

Figure 6.2: Threshold-based filter

### 6.1.1 Static Threshold-based Strategy

The simplest filter condition is based on a threshold that was also used in the example above. The threshold-based static filter $filter_{threshold}$ sets a comparison matrix entry to false, if a similarity value $sim(x, y)$ of an element pair in the input similarity matrix is smaller than a given threshold. In the example from Figure 6.2 the resulting comparison matrix after filtering is shown. In a potentially following matcher operation, $s1$ would be compared to all target elements whereas $s2$ would only be compared to $t1$. As with static thresholds for the selection operator a single threshold for all pairs of elements can be problematic. The absolute similarity values often do not give enough evidence for filtering and selection.

### 6.1.2 Relative Threshold-based Strategy

The threshold of $filter_{threshold}$ can also be formulated as a threshold relative to the maximum achievable similarity values for a schema element as was similarly done with maximum-based selection in Section 2.4.2. Figure 6.3 illustrates the behavior of the relative threshold-based filter $filter_{relative}$ . The relative Forward and Backward similarity matrices are computed as described by Melnik et al. [117] within their Perfectionist Egalitarian Polygamy. The threshold filters pairs with low similarity. Finally, all pairs that remain part of the selected similarity matrices will get a true entry in the comparison matrix.

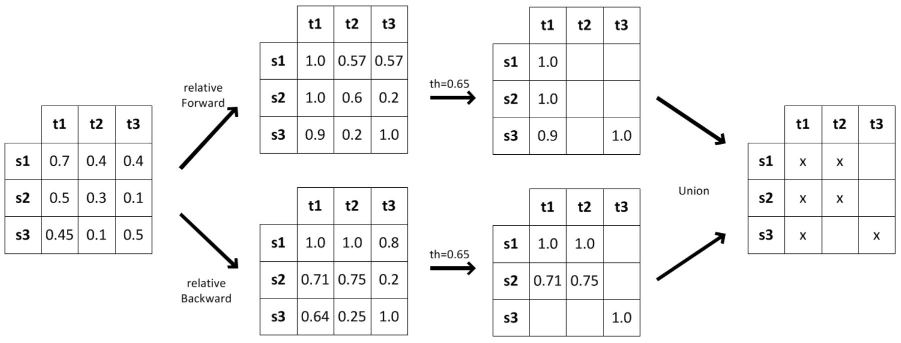


Figure 6.3: Relative threshold-based filter

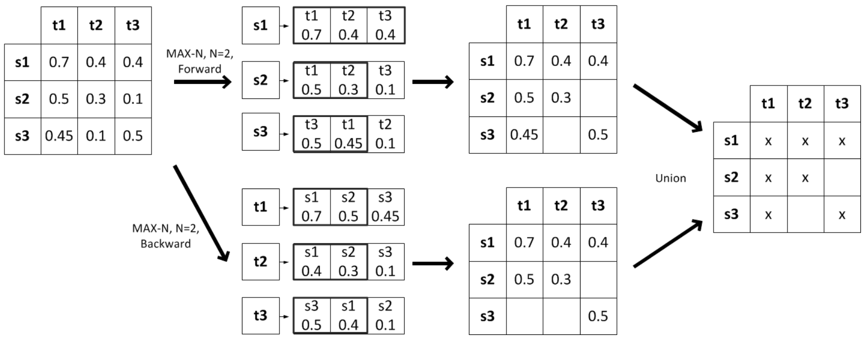


Figure 6.4: TopN $N = 2$ filter strategy

### 6.1.3 Static TopN Strategy

Similar to the relative threshold, a filter could restrict the number of comparisons based on a top-N condition. This ensures that each element is compared to the best N candidates from the first matcher in a sequential process. The TopN filter strategy $filter_{topN}$ relies on a MAX-N both selection that selects the N best pairs of a similarity matrix from a Forward and Backward selection. All identified pairs are part of the resulting comparison matrix. Figure 6.4 illustrates the behavior of the TopN filter with N=2. The resulting comparison matrix again shows a good distribution of comparisons. Each element will be compared with at least two elements in the source or in the target. For comparison filtering in a real world setting higher N values (e.g. N=20) need to be used to not prune pairs that are potentially correct correspondences. Also the N value should be higher for larger mapping problems.

### 6.1.4 Dynamic Threshold-based Strategy

Up to now, all presented filter strategies could prune element pairs that are potentially candidates of the matching process without filtering. Thus, by filtering with the presented strategies the recall could be reduced. The dynamic filter strategy that is presented below is different. If adapts itself to the already processed comparisons and mapping results. Recall that many matching processes consist of a number of Match operators, a Combine operator and a Select operator as shown in Figure 6.5 (a). The idea of a dynamic threshold strategy $filter_{dyn-threshold}$ is to filter out comparisons that already lost its chance to survive the final selection. Assume, a process consists of a set of Match operators $\{m_1, ..., m_n\}$ that are contributing to a single Combine operator $C$ and Select operator $S$. Each matcher $m_a \in \{m_1, ..., m_n\}$ has a weight $w_a$ and computes a result similarity value $sim_a(s,t)$ for each element pair $(s,t)$. The combination operator $C$ applies the WEIGHTED combination strategy from Section 2.3.2 and as selection strategy the THRESHOLD strategy $select_{threshold}$ is used. The chance of not being pruned out can be computed after a matcher $m_x$ has been executed. Given the threshold $th$ of the final selection $S$ the following condition can be evaluated:

$$\frac{\left(\sum_{\{m_1 \dots m_n\}/m_x} w_m * sim_m(s,t)\right) + w_x * sim_x(s,t)}{\sum_{k=1 \dots n} w_k} < th \qquad (6.1)$$

If a matcher is not yet executed, the maximal possible similarity $sim_m(s,t) = 1$ is taken. If the computed combined similarity is smaller than $th$ then the comparison can be pruned by setting the respective cell in the comparison matrix to false. When more matchers are already executed, the actual similarities of matchers $sim_m(s,t)$ are known so that it will be much more probable that an element pair will be pruned. The dynamic filter condition ensures that the result of a filtered execution will not differ from a parallel execution. However, in most cases the dynamic filter does only begin pruning element pairs after some matchers have been executed.

Example: Imagine three matchers with weights $w_1 = 0.3$, $w_2 = 0.4$ and $w_3 = 0.3$ that contribute to a WEIGHTED combination strategy and a following THRESHOLD selection with $th = 0.7$. If the first matcher computes a similarity for two elements $sim_1(s,t) = 0.2$ then the dynamic filter will not prune the comparison $((0.4 * 1 + 0.3 * 1) + 0.2 * 0.3) = 0.76 > 0.7$. The more matchers are involved, the more unlikely it is that an element pair will be pruned early on. If the second matcher results in $sim_2(s,t) = 0.35$ then the element pair can be pruned since it will not survive the selection. $((0.4 * 0.35 + 0.3 * 1) + 0.2 * 0.3) = 0.5 < 0.7$. This dynamic strategy can be softened by setting the worst case result similarities smaller than 1: $sim_m(s,t) < 1$ for matchers that have not yet been executed. However, this could again reduce recall.

The $filter_{dyn-threshold}$ strategy makes some assumptions about the structure of a matching process of having a number of parallel matchers, a WEIGHTED combination and a selection operator at the end. However, the approach could also be applied (with adaptations) in cases where other combination strategies and matchers and selections are applied in sequence. The dyamic filter operator would then have to pre-compute the chance of surviving a final selection by pre-computing all operators on the path to the final selection.

### 6.1.5 Dynamic Delta-based Strategy

The $filter_{dyn-threshold}$ strategy is interesting since it filters comparisons without changing the quality of the matching process. However, if many matchers are involved, the filtering begins rather late. Also, the dynamic threshold-based filter can only be applied if a THRESHOLD-strategy is used for selection and a combination of THRESHOLD and MAX-N or MAX-DELTA strategies are much more commonly used since they typically achieve better quality. Therefore a $filter_{dyn-maxdelta}$ strategy is proposed. The strategy collects for each source and target element the maximal values in each column and row. These arrays $maxRow$ and $maxColumn$ are initially empty and are of size $|S|$ and $|T|$ respectively. With each newly incoming similarity value $sim(s,t)$ the maximum of the column of $t$ and the maximum of the row of $s$ can be updated. Again, assuming a process consists of a set of Match operators $\{m_1, ..., m_n\}$, a Combine operator $C$ with WEIGHTED strategy and Select operator $S$ with MAX-DELTA strategy. Each matcher $m_a \in \{m_1, ..., m_n\}$ has a weight $w_a$ and computes a result similarity value $sim_a(s,t)$ for each element pair $(s,t)$. The chance of not being pruned out can be computed after a matcher $m_x$ has been executed. A projected combined similarity value $pSim$ can be computed as above:

$$pSim = \frac{\left(\sum_{\{m_1 \ldots m_n\}/m_x} w_m * sim_m(s,t)\right) + w_x * sim_x(s,t)}{\sum_{k=1 \ldots n} w_k} \tag{6.2}$$

Given the $delta$ value of the final selection $S$ the following condition can be evaluated:

$$\begin{aligned} pSim \leq maxRow_s - delta * maxRow_s \vee \\ pSim \leq maxColumn_t - delta * maxColumn_t \end{aligned} \tag{6.3}$$

The more conditions are evaluated the higher the maximum values in the arrays $maxRow$ and $maxColumn$ get. Thus, the probability for a pair of being pruned increases with execution.

Example: Imagine two matchers with weights $w_1 = 0.7$, $w_2 = 0.3$ that contribute to a WEIGHTED combination strategy and a following MAX-DELTA selection with $d = 0.1$. If the first matcher computes a similarity for two elements $sim_1(s_1, t_1) = 0.7$ and $maxRows_1$ as well as $maxColumn_1$ are empty nothing is filtered. The projected similarity after the combination is $pSim = (0.7 * 0.8) + (0.3 * 1) = 0.86$. The value for $maxRows_1$ is computed as $(0.7 * 0.8) + (0.3 * 0) = 0.56$. It assumes that the second matcher will return a similarity value of 0. If the first matcher then computes a similarity for two elements $sim_1(s_1, t_2) = 0.1$ then $pSim = 0.7 * 0.1 + (0.3 * 1) = 0.37$. The value is smaller than $maxRows_1 - maxRows_1 * 0.1 = 0.504$ so that the comparison can be prunded for the second matcher.

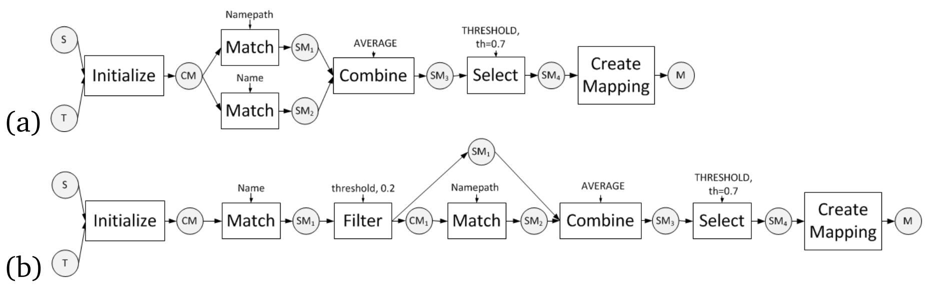


Figure 6.5: Parallel and sequential process

## 6.2 Evaluating Sequential Matcher Combinations

With the Filter operator, sequential matching processes can be constructed that potentially perform much faster than their parallel equivalent. Figure 6.5(b) shows a matching process that executes a Name matcher and a Namepath matcher in sequence. Executing a label-based matcher before executing a structural matcher is quite common in sequential matching systems as was discussed in Section 3.1.2. We evaluated the quality and performance of both processes with different pairs of matchers from a small library [134]. The library contained the following matchers: Name (NM), Namepath (NPM), Children (CM), Parent (PM), Data-type (DTM) and Leaf (LM). For all pair-wise combinations of these matchers a parallel combined and a sequential combined process was created (see Figure 6.5(a) and (b)). For evaluation, the purchase order dataset from the COMA evaluations was used. In order to be comparable, the selection thresholds, the filter thresholds and combination weights for both processes where tuned automatically to achieve the best quality possible. A huge space of parameter settings was tested in high detail with a brute-force approach. From the best performing configuration the fastest ones where taken and analyzed. The results can be found in Figure 6.6. It shows average execution times for matching a set of schemas with a combined and a sequential process using the matcher pair on the x-axis. In a number of cases the sequential process performed faster than its combined equivalent. However, in some cases the sequential processes performed slower. The reason is that the best quality configuration for the sequential and combined process was compared. For some pairs the filter threshold had to be 0 since any comparison filtering would prune correct correspondences. The achieved f-measure of such sequential processes is equal to the combined equivalent. However, they perform slower than the combined ones since they have to cope with the additional overhead of the Filter operator. Yet the majority of combinations used a filter threshold that was bigger than 0. The execution times of those sequential processes are significantly smaller than the combined ones. In some cases a significant part of comparisons was dropped out after the first matcher executed. Obviously there are some matchers that have better “filter-properties” whereas others should not be used as filter matcher.

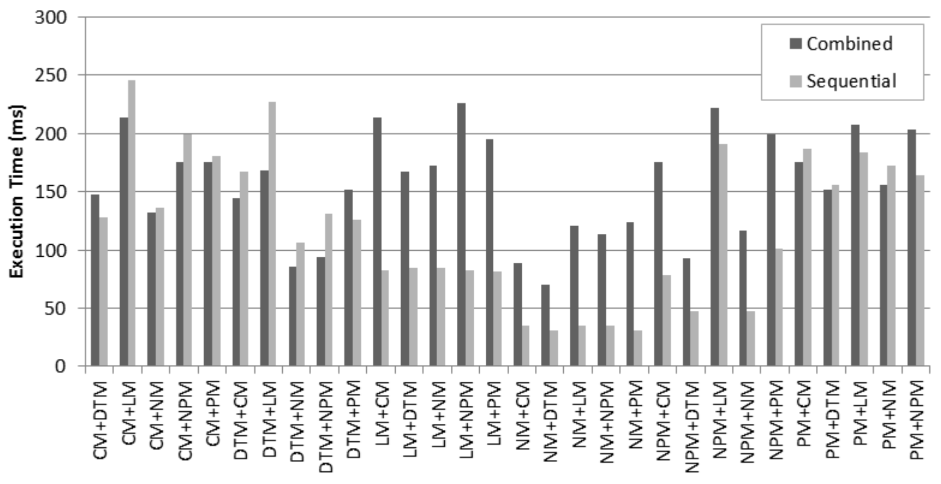


Figure 6.6: Comparing execution time of best strategies

The goal in the following sections is to automatically transform a parallel combined matching process into a sequential process without losing quality. For that purpose a formal representation (the Incidence Graph) of the pairwise evaluation results is introduced. Based on this Incidence Graph rules are constructed that automatically rewrite a parallel matching process.

### 6.2.1 Incidence Graph

In order to make the observations of sequential matcher pairs reusable a so-called Incidence Graph is introduced. Such graph represents the well-performing combinations as a graph data structure.

**Definition 15.** ***(Incidence Graph)*** *The Incidence Graph is a directed graph that describes incidence relations between matchers $m$ from a matcher library $\mathcal{M}$ for a given set of mapping problems. The graph vertices represent matchers. The mapping problems are executed with a parallel matching process and its sequentially filtered equivalent. If a significant average speedup ($> 20\%$) can be achieved with sequentially executing matcher $m_x$ and matcher $m_y$ an edge $(m_x, m_y)$ is added to the graph. Each matcher $m_i$ is annotated with the average time $R_i$ to execute the matcher on the given set of mapping problems. Edges are annotated with the filter threshold that was found for the filter operator in the sequential execution example. They also store the average percentage of the achieved speedup $P_{xy}$ when executing matcher $m_x$ before matcher $m_y$. $P_{xy}$ can be computed as follows: $P_{xy} = 1 - (R_{seq}/R_{comb})$ with $R_{seq}$ being the run-time of the sequential process on the given mapping problems and $R_{comb}$ being the run-time for the combined process. The higher the value $P_{xy}$ is the better the sequential speedup is.*

Sometimes two edges $(m_x, m_y)$ and $(m_y, m_x)$ between two matchers $m_x$ and $m_y$ are put into the graph. This happens if two matchers behave similarly and therefore

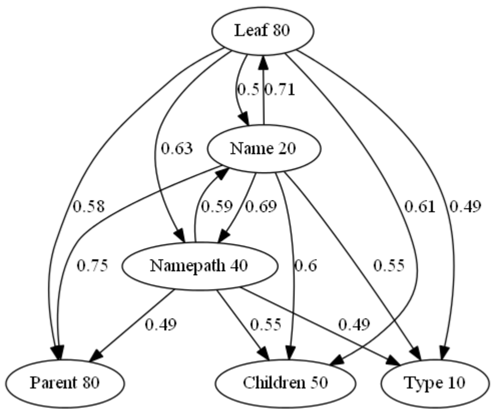


Figure 6.7: Incidence Graph example containing only individual run-times and the achieved speedup

serve as good filter matchers for one another. Figure 6 shows the computed graph for the given matcher library. For simplicity the found filter thresholds are omitted. An edge from the Name matcher to the Leaf matcher states the following: The run-time of sequentially executing the Name matcher before the Leaf matcher was 71% faster than the parallel combination of these matchers ($P_{xy} = 0.71$). The average run-time $R_x$ of the individual matchers on the given mapping problem is associated to the corresponding node. It can be observed that the combinations that are encoded in the graph are quite stable for a given matcher library and different mapping problems. Also the computed filter thresholds that do not deteriorate quality were similar. However, the graph should be recomputed for each new matcher library and problem domain since these properties are not generalizable. The information about the achieved relative speedup and the individual run-time of matchers is later used to decide, which sequential combination of two matchers is the best. The information in the graph will be an integral part of a simple cost model when automatically changing matching processes.

## 6.3 Matching Process Rewrite Technique

As said, parallel matching processes shall be rewritten to its sequential equivalent to improve performance. For that purpose a novel graph-based rewrite technique is presented. The rewrite technique generalizes any process changes with the help of rewrite rules. Given a matching process, the decision of change could depend on the current process structure and properties of individual operators. A matching process rewrite rule is defined as follows:

**Definition 16.** ***(Matching Process Rewrite Rule)*** *A matching process rewrite rule describes how to change a matching process. It consists of three parts:*

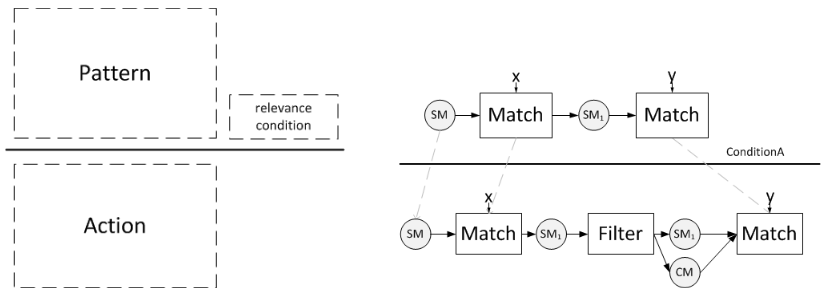


Figure 6.8: Rule notation (left) and example matching rule (right)

- *A process pattern $p$*
- *A relevance condition $r$*
- *An action $a$*

*The pattern $p$ describes parts of a process where the rule can be applied to. The relevance condition $r$ defines when the respective rule should be applied for a found pattern instance $p$. It can be used to decide whether a rule should be executed or not. This decision can be based on some external knowledge like execution times of operators. The action $a$ describes how to change instances of the found pattern p. This includes additions and changes of one or many (additional) operators to a process. It is represented as a process structure with references to the original pattern $p$ but could also be extended by a function that computes the new process structure and necessary parameters.*

Later, in Chapter 7 this matching process rewrite rule definition is extended so that the relevance is also computed from features of intermediate results or the input schemas to support adaptivity.

A simple notation for illustrating matching process rewrite rules will be used within the following chapters. The notation containing pattern, condition and applied change is shown on the left in Figure 6.8. If the condition evaluated over a found pattern instance is true, the changes below the horizontal bar are applied to the pattern instance. On the right of Figure 6.8 a sample rule is shown. The rule describes a pattern of two succeeding matchers x and y within a given process graph. When the condition $conditionA$ evaluates to true, it adds a filter operator in between all found pairs of these matchers to reduce the search space for matcher y. In this chapter we focus on filter-based rewrite rules. Yet, a number of other rules involving other schema and mapping operators are feasible.

Based on rules, rewrite-based process optimization can now be defined:

**Definition 17.** ***(Matching Process Rewrite Technique)***
*Given:*

- *A matching process $MP$ as defined in Section 4.1.*

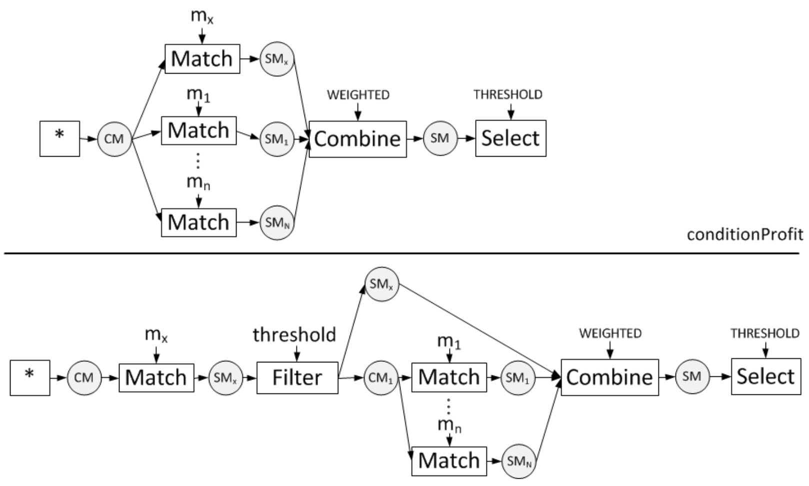


Figure 6.9: Filter-based rewrite rule ($Rule_{threshold}$)

- *A set of Rewrite Rules $RW$ that transform a matching Process $MP$ into a rewritten matching process $MP'$*

*The goal of rewrite-based matching process optimization is to create a new matching process $MP'$ by applying rule $rw \in RW$ onto a matching process $MP$ written as $rw(MP) = MP'$ so that the performance and/or the result quality of a matching process improves.*

*In this Chapter a few rewrite rules are introduced that improve the performance of a matching process. In Chapter 7, rewrite rules will also focus on improving quality.*

### 6.3.1 Filter-based rewrite rules

The example rule from above adds a filter operator in between a found pair of matchers to reduce the search space for the second matcher. The rule can be generalized to more complex patterns involving several matchers. The observation from above that can be utilized is that matching processes are typically finished by selections to select correspondences exceeding a certain similarity threshold as likely match candidates. From that knowledge, a rewrite rule for matching processes that consist of parallel matchers $m_x, m_1, ..., m_n$ can be created. The rule relies on a relevance condition $conditionProfit$ that utilizes a relation $<_{profit}$ over a set of matchers $\mathcal{M}$. A set of matchers $\mathcal{M}_p = \{m_1, ..., m_n\}$ profits from a matcher $m_x \notin \mathcal{M}_p$ written as $\mathcal{M}_p <_{profit} m_x$ if the following holds: There is a an edge in the incidence graph from $m_x$ to each $m_i \in \mathcal{M}_p$. Based on that relation a filter-based rewrite rule $Rule_{threshold}$ can now be defined as shown in Figure 6.9. The condition is defined as $conditionProfit = \{m_1, ..., m_n\} <_{profit} m_x$.

An operator with * references any operator that provides similarity matrices and/or comparison matrices. The pattern on top of the bar describes a part of a

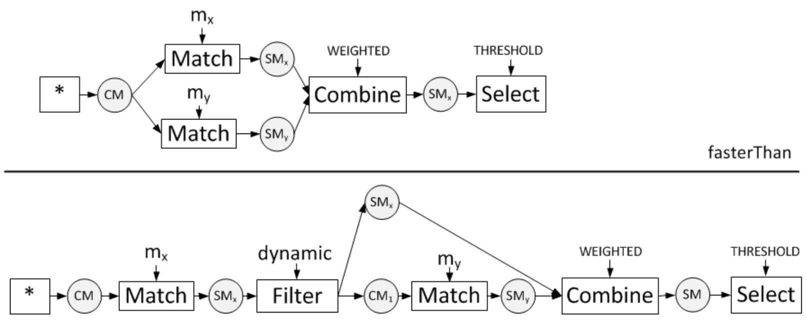


Figure 6.10: Filter-based rewrite rule ($Rule_{dyn}$)

matching process that consists of a set of matchers $\{m_1, ..., m_n\}$ plus a matcher $m_x$ that is executed in parallel. Their result is combined with a WEIGHTED combination and selected with a THRESHOLD strategy. The $conditionProfit$ evaluates to true, if a set of matchers profits from $m_x$ written as $\{m_1, ..., m_n\} <_{profit} m_x$. The applied rewrite below the bar adds a filter operator after matcher $m_x$. The used filter strategy is $filter_{threshold}$ with a static threshold. Alternatively also one of the other filter strategies can be used. The input of all matchers $\{m_1, ..., m_n\}$ will be changed to $CM_1$. The filtered result of $m_x$ will be added to the combination. All matchers $m_i \in \{m_1, ..., m_n\}$ that do not profit from $m_x$ remain unchanged.

The use of filter-based rewrite rules is analogous to the use of predicate push-down rules for database query optimization which reduce the number of input tuples for joins and other expensive operators. The filter-based strategy tries to reduce the number of element pairs for match processing to also speed up the execution time. In particular, rewrite rules could lead to changes in the execution result of a matching process while database query optimization leaves the query results unchanged. Each Filter operator in a matching process could get different threshold-values that are adapted to the mapping they need to filter. When the rewrite rule is applied, the annotated incidence graph that stores a filter threshold for each edge is reused. If there are multiple outgoing edges from matcher $m_x$ a defensive strategy is applied: The threshold is set to the smallest threshold of all outgoing edges in the incidence graph from $m_x$ to matchers of $\{m_1, ..., m_n\}$. Since applying the dynamic filter condition can be done between arbitrary matchers without changing the final result a further rewrite rule $Rule_{dyn-threshold}$ is introduced (see Figure 6.10). Whenever two matchers $m_x$ and $m_y$ are executed in parallel, these matchers are applied in sequence and a Filter operator is put in between them with a dynamic filter such as $filter_{dyn-threshold}$. The $fasterThan$ condition defines that the execution time $R_x$ of the first matcher is smaller than the execution time $R_y$ of the second matcher.

Similarly a rule that relies on $filter_{dyn-maxdelta}$ can be defined that is called $Rule_{dyn-maxdelta}$. Typically dynamic rewrite rules will be applied after $Rule_{threshold}$ has already been executed.

### 6.3.2 Applying Filter-based Rules

To apply matching process rewrite rules for filtering an algorithm is needed that is able to find suitable patterns for adaptation in order to improve performance. The algorithm *applyRule* (see Listing 6.1) takes an incidence graph IG, a matching process MP and a specific rewrite rule R as input. Initially, all matching pattern

**Algorithm 6.1** ApplyRule

```
 1  ApplyRule(IncidenceGraph IG, MatchingProcess MP, Rewrite Rule R)
 2    MPNew ← MP
 3    patInstances ← findPatterns(R,MP)
 4    costMap ← ∅
 5    FOR EACH p in patInstances
 6      cost ← computeCost(p,IG,R)
 7      costMap.put(p,cost)
 8    IF costMap.size > 0
 9      best ← costMap.minimum
10      MPNew ← rewrite(best,R)
11      MPNew ← applyRule(MPNew,IG,R)
12  END
```

instances are identified in the given process (line 3). In line 6, for each pattern instance the cost C is computed as described in the following paragraph. The cost estimates are stored in a map (line 7). If the costMap is not empty, the best pattern instance is selected in line 9 and rewritten in line 10. The function applyRule will be called recursively in line 11 in order to iteratively rewrite all occurrences of the given pattern. The algorithm terminates when the costMap is empty and all possible pattern instances are rewritten (see line 8).

A simple cost model based on the incidence graph is used to decide which pattern instance to rewrite for $Rule_{static}$ and $Rule_{dyn}$.

Given:

- The incidence graph that contains individual run-times of matchers $R_x$ for all matchers $m_x \in \mathcal{M}$.
- The percentage of relative speedup $P_{ab}$ between two matchers $m_a$ and $m_b$ as defined above. If there is no edge in the incidence graph from $m_a$ to $m_b$ then $P_{ab} = 0$.

The cost $C_{x,\{1..n\}}$ of executing matcher $m_x$ before a set of matchers $\{m_1, ..., m_n\}$ can be computed by:

$$C_{x,\{1..n\}} = R_x + \sum_{a=1,..,n} (1 - P_{xa}) * R_a \tag{6.4}$$

The rationale behind this cost-model is the following: The first matcher $m_x$ must be executed, hence its full run-time $R_x$ is considered. All matchers that have an incoming edge from $m_x$ add a fraction of their run-time cost to the overall cost that

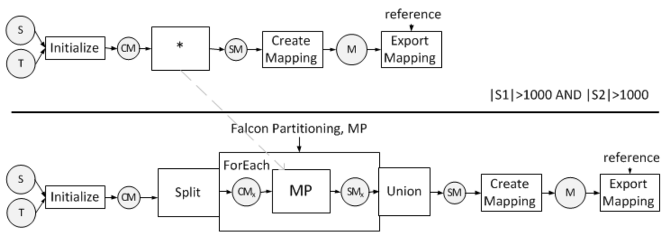


Figure 6.11: Falcon Partitioning rule

depends on the anticipated relative speedup $P_{ab}$. Computing the cost of a parallel execution of the given matchers is straightforward. Only the run-time costs of all individual matchers need to be summed up.

Example: Taking the values from the example incidence graph above the computed cost for first executing the Name matcher and then executing all other matchers is: $20+((1-0.55)*10)+((1-0.6)*50)+((1-0.71)*80)+((1-0.75)*80)+((1-0.69)*40) = 100.1$. Whereas first executing the Namepath matcher would generate higher cost: $40+((1-0.49)*10)+((1-0.59)*20)+((1-0.55)*50)+((1-0.49)*80)+80 = 196.6$.

### 6.3.3 Further Rules

Further rules can be defined that increase the performance of a matching process. For instance, with the rewrite approach a partitioning technique as it was introduced within Coma [142] or Falcon [77] could be expressed as a rewrite rule. If the schema size exceeds a certain threshold a given matching process could be wrapped into a For-Each operator. The rule is shown in Figure 6.11. The main purpose of the partitioning in Falcon was to compute smaller mapping problems that are easier to compute with restricted memory. The approach also prunes comparisons of computed blocks which could increase performance on bigger mapping tasks.

Another area where performance of matching processes can be increased is to avoid repeated executions of the same matchers. This happens if the same operator with equal parameters is used multiple times within a single process. However, only in specific cases this can be done. If two Match operators with equal matching strategy do have no input mapping or equal input mappings and equal input comparison matrices, then one of the operators can safely be replaced. If the two matching operators have different input comparison matrices more advanced rewrites would be needed. Three cases could occur: The comparison sets could overlap, one set could be fully included in the other set or two operators have no comparisons in common. In Figure 6.12 (left) a rule is shown that rewrites the process to reuse one Match result. If a comparison matrix CM is treated as a set of comparisons then the condition can be formulated as follows: $ConditionX = CM_x \subset CM_y$. For reusing a match result a simple matching strategy is introduced that is called “None”. The matching strategy simply takes the similarity values from the input constituent

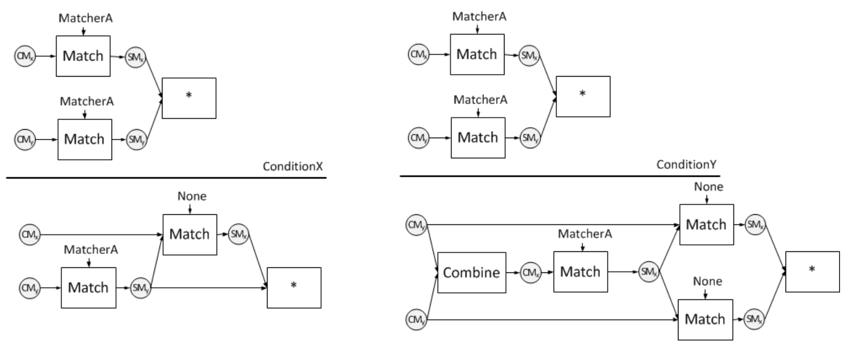


Figure 6.12: Reusing Match operators with differing comparison matrix input

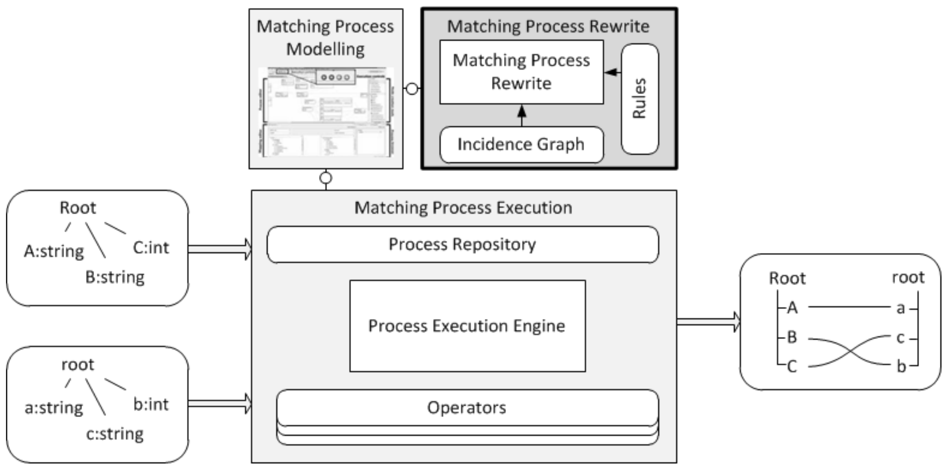


Figure 6.13: System architecture

similarity matrix and takes them as output values if the respective comparison matrix entry is set to true. The rule could be extended to also cope with cases where the comparison sets only intersect. An appropriate condition $ConditionY$ can be formulated as $(CM_x \not\subset CM_y) \wedge (CM_y \not\subset CM_x) \wedge (CM_x \cap CM_y \neq \emptyset)$. The resulting rule is shown in Figure 6.12. If the intersection of comparisons is very small, the additional overhead of combining comparisons and additional Match operators with the None matcher strategy would not pay off.

### 6.3.4 Matching Process Rewrite System

Based on the concepts of rewrite-based matching process optimization a new component for rewriting matching processes was developed. The architecture in Figure 6.13 illustrates how the new rewrite-based approach embeds itself into the components that were presented until now.

The matching process modeling tool that was presented in Chapter 5 directly interacts with the new Matching Process Rewrite component. Matching processes are defined at design time by using the operators from the matching process model. While modeling, the designer could ask the rewrite system to tune the performance of his process. Automatic rewrites can be confirmed or rejected by the user. The rewritten graph can then be further extended or changed before finally storing it in the process repository.

## 6.4 Advances over Related Work

In comparison to all existing approaches that were presented in Section 3.4 for improving the performance of a matching system, the presented approach is the first one that tries to generically improve performance on the process level. Analogous to cost-based rewrites in database query optimization, the performance improvement problem is treated as a rewrite problem on matching processes. This allows expressing performance and quality improvement techniques by special rewrite rules. Note that the rewrite approach is orthogonal to existing techniques for improving the performance of matching systems. The concept and some filter rules could already be published as a conference paper [134]. Within this chapter, the definitions of rewrite rules were refined and additional filter strategies and rewrite rules were presented. In particular the dynamic filter strategy that can be used with MAX-DELTA selections is a major step forward since many well-performing matching systems rely on MAX-DELTA instead of a simple THRESHOLD strategy.

Still, some issues remain. The incidence graph is computed in a training phase, thus correct reference mappings are needed that are similar to the mapping problems to be solved. The simple cost model for choosing rules only relies on pairs of matchers. Certainly, longer sequences of matchers can be constructed and the cost for choosing rewrite rules could differ from the measured costs in these sequences. And finally, the order of applying rewrite rules must be defined. Currently, rules are given priority, so that first the static filter rules are applied before applying the dynamic rules.

As can be shown later in the Evaluation (Section 8.2), the rewrite-based approach to increase performance is effective. In the next chapter the rewrite rule approach is extended to also improve the quality of a matching system.

# Chapter 7

# Adaptive Schema Matching based on Rules

This chapter introduces an approach that automatically selects and executes rewrite rules when matching two input schemas to increase matching quality. A preliminary version of the approach could already be published on the ICDE conference [136]. Based on the presented rewrite-approach from Chapter 6, a matching process is created adaptively. A set of rules is collected and exemplary executions of the adaptive process are described. The chapter closes with a discussion of the presented contributions and a comparison to similar or related approaches.

## 7.1 Rule Definition

Recall from the previous chapters that the choice of matchers, combination and selection strategies that can be applied for solving a matching problem is big. In addition, different matching process patterns can be combined which leads to a huge space of possible matching process configurations. Currently, the internal matching processes of matching systems are mostly built manually based on experience. When to apply a pattern or operator is mostly implicit matching knowledge. For instance, iterations and similarity propagation should only be applied for mapping problems with high structural similarity of the input schemas and in cases where basic matchers are not able to identify many seed matches. The skimming or refinement patterns are valuable, but should not be used for all possible mapping problems. It can be observed that such matching knowledge is often hidden in code within research prototypes. The matching community tends to rebuild whole matching systems for implementing and evaluating a minor improvement to a specific operator. Reusing algorithms and matching knowledge from such black box matching system implementations is complex and often impossible. It can also be observed that many authors of new matching, selection or combination algorithms analyze in their evaluations for what types of schema mapping problems their algorithms are appropriate and when they should not be applied. Sometimes, also the relation to schema features is explicitly given in textual form. But still, the textual representation is often vague and the features of the mapping problems are often not quantified.

Obviously, a standardized representation of matching knowledge would be valuable. With the help of the Condition construct and the presented features from Chapter 4, such matching knowledge can be represented. However, the more knowledge about when to use matchers, selections, combinations or patterns are included in a manually modeled matching process the more complex a process gets. Clearly, it is not feasible to build an allround matching process that contains all kinds of matching knowledge. It would be desirable to encode matching knowledge into smaller building blocks.

Surprisingly, the matching process rewrite rules from Chapter 6 with minor extensions do serve well for that purpose. Recall that a process rule consists of a pattern $p$, a relevance condition $r$ and an action $a$. The relevance condition $r$ can be based on features that are computed from the input schemas and intermediate results of executing an input matching process. The action $a$ of such rules describes how to change found instances of a given pattern $p$ in a matching process. This can be extended by additional functions that compute parameters of the changed process or the process structure. These additions allow capturing design decisions a matching process expert would take to increase the quality and performance of a matching process for a given mapping problem. A new matching system is proposed that is able to execute and combine rules adaptively based on the features of the input schemas and on intermediate results of executing process parts. This adaptive process is similar to what a matching expert would do to build a matching system manually by using the proposed tools from Chapter 5. A process is constructed, then executed and intermediate results are analyzed. Based on that analysis, the process is changed.

## 7.2 Adaptive Process Construction

The general structure of a matching rule enables users to define arbitrary rules with complex patterns, conditions and actions. A single rule could add single operators or create a complete matching process. The question is how to combine rules automatically to adaptively build a matching process. The naive approach of letting all rules compete to be executed is not feasible. This leads to problems of order and missing control. Also, termination could not be ensured. In his diploma thesis that was run in the context of this doctoral work, Julian Eberius [48] was able to identify an initial set of complex rules that were combined within a stepwise process. Due to the complexity of rules that can add and remove multiple matchers to a process, side-effects of rule application occured. It could be learned, that rules need to be simplified and the rule selection must be strongly controlled.

In order to restrict the complexity of rule choice and execution only a small set of rule types is introduced in the proposed adaptive matching system. Moreover, the application of these rules is only performed within a fixed number of stages. In the following, the general workflow of executing rules in stages is introduced before the rule types are described in detail.

### 7.2.1 Staged Execution

As a starting point the order of executing matching operators within existing matching systems was analyzed. An often used approach is to first execute basic matchers, combine their results, then execute structural matchers and finally select most promising correspondences to compute a mapping. This typical approach is generalized as five stages of rule application that are Initialize, Refine, Combine, Rewrite and Selection. In each stage all used matching operators are chosen adaptively. This also includes the Combine and Select operators and their strategies. Moreover, the Refine, Combine and Rewrite stages are executed iteratively.

The stage-wise execution of rules is illustrated in Figure 7.1. On the right, the construction and stage-wise change of the matching process is shown schematically. The most recently added matching operators are grey boxes. The process starts with

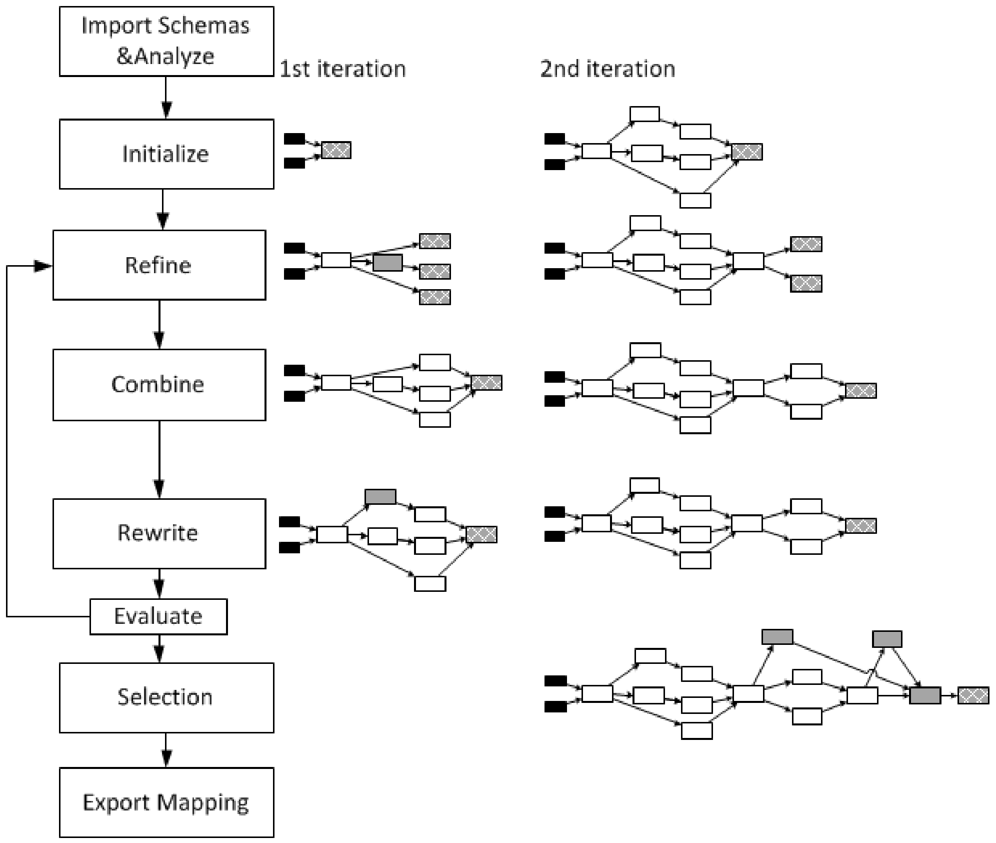


Figure 7.1: Stagewise execution of rules

the Initialize stage that creates an initial empty matching process with operators for importing the source and target schema and an operator that initializes a mapping (Initialize operator). In the following Refine and Combine stages the process is extended at the nodes that do not have any following operators. Such operators are called *dangling operators*. Dangling operators are marked with a white cross pattern in Figure 7.1. The Initialize stage only creates one dangling operator. In the Refine stage so-called refine rules add matching operators or other process substructures to the latest dangling node. Each refine rule could create one or more new dangling nodes for the next stage. In the first iteration they mostly add basic matchers whereas in later iterations they could add structural matchers that rely on a similarity matrix as input.

In Figure 7.1 three dangling operators were created in the Refine stage. Note, that the rules that add basic matchers and structural matchers are subsumed into one stage since a basic matcher also refines the current intermediate result which is empty in most cases. However, the system could also start with a given user defined input mapping so that a first refinement could also execute structural matching operators. The following Combine stage adds operators that combine all dangling operators from the Refine stage to a single operator. In each Combine stage only one so-called combination rule can be executed.

Afterwards, in the Rewrite stage the newly created operators from the current iteration can be rewritten without changing the dangling operators. Operators can be reordered, new ones can be added or parameters of operators can be changed. The changes of the Rewrite stages are executed and evaluated using matrix features that project mapping quality. No operator is executed twice and only newly added or changed parts are executed. For evaluating the quality of the most recent iteration, the Monogamy feature is used. If the evaluation shows improvement of quality, then a new iteration is started that again executes Refine, Combine and Rewrite stages. In the schematic example process the second iteration adds two dangling operators in the Refine stage, combines them and does not perform any rewrites. If an evaluation does not identify improvements in the Monogamy the Selection stage is started. If the projected mapping quality measured by Monogamy does not increase then the effects of the most recent iteration can be ignored and therefore reverted.

The Selection stage executes selection rules that add Select operators to the most recent dangling node. Each selection rule appends a new dangling node. Note that also the dangling nodes from each iteration can be included in the selection process. This allows Selection rules to implement the skimming pattern. Finally, a Create-Mapping operator is added to the last Select operator, all the newly added operators are executed and a computed mapping is returned.

Obviously each stage executes a different type of rule. Rules rely on the process that was created in a previous stage. Four types of rules can be distinguished, each belonging to an execution stage.

**Refine rules** extend an input process by adding new operators to the dangling nodes of the process. These added operators should increase result quality. This can be achieved by adding Match operators with new matching techniques that exploit not yet processed schema information. However, if a particular matching technique adds misleading evidence the result quality could also decrease. The decision of applying a particular refine rule depends on schema features but also on the features of intermediate execution results of a matching process. For instance, some refine rules add structural matchers to a matching process to propagate already found similarity values and identify additional structural matches. Propagation exploits the fact that two elements are similar if their neighboring elements are also similar. However, this evidence of neighboring element similarity does only help if the two schemas to be matched are also structurally similar. That structural similarity can be measured by features as described above.

**Combination rules** add Combine operators to a process and combine dangling operators from a matching process. Dangling operators have been created by refine rules. A multitude of combination strategies were described in Section 2.3 each with advantages and disadvantages. However, also trees of Combine operators could be created to combine the dangling operator results. The problem of combination is to assign weights to the different dangling outputs from the Refine stage. In order to find the most appropriate combination strategy matrix similarity features can be used within the relevance conditions of combination rules.

**Rewrite rules** take a non-empty matching process that was created by refine and combination rules as input and rewrite the process to a new matching process MP'. Rewrite rules change the structure of a given process without changing dangling output operators. For instance, the order of operators could be changed or additional operations such as Filters can be added in between others.

**Selection rules** are applied before finalizing a matching process. They are used to add Select operators to the last dangling node of the current matching process. As with the Combine operator, a number of selection strategies were presented in Section 2.4 that need to be chosen adaptively based on features of the schemas and intermediate results.

Due to the different predefined stages, termination can be ensured and side-effects of rule execution can be minimized. In the Refine stage, individual rules only add operators to the process. Such rules cannot influence the effects of other rules. In the Combine stage only one combination rule is executed. The only remaining problem is to choose the most appropriate rule for a given set of intermediate results from a Refine stage. The Rewrite stage is more critical. Rules could potentially influence each other so that the order of execution could make a difference. However, each rule can only be executed once which ensures termination. The order of how rewrite rules can be applied is predefined and can for instance be defined based on the number of changes a rule performs. This ordering or prioritization is a typical approach to cope with side-effects of rewrite rule execution which is similar to graph transformation rules [33]. Finally, in the Selection stage each selection rule can only be executed once. Since selection rules could also append operators to existing operators created by other selection rules the order of rule execution matters. Therefore selection rules are ordered on how strong they filter matching results. The most restrictive selection rules should be performed latest if they are relevant.

## 7.2.2 Executing a Stage

Within each stage of the process a predefined rule execution process is started (see Algorithm 7.1 ). The ExecuteStage algorithm gets as input the name of the current

**Algorithm 7.1** ExecuteStage

```
1  ExecuteStage(Stage s, Rules rset, MaxCount N, MatchingProcess MP)
2    d <- collectDangling()
3    i <- 0
4    FOR EACH r in rset
5      IF i>=N return;
6      IF r.matchesPattern(d,MP) AND r.relevant(d,MP)
7        r.apply(d)
8        exectuteStepping(MP)
9        i++
10   END
11 END
```

stage, the set of rules for the given stage and the maximal number of rules that should be executed. The process begins with collecting all current dangling nodes (Line 2). Then, for each rule its rule pattern is matched to the matching process and the relevance condition is evaluated (Line 6). If a rule is relevant, it is applied to the matching process. Applying a rule implies changes to the current matching process. The Combine stage only allows for one rule execution so that N is 1. All other rules are applied until N rules were executed.

Note that it must be ensured that there is at least one selection and one combination rule as fallback that rate itself as relevant. The fallback combination rule should add a Combine operator with an AVERAGE strategy and the fallback selection rule should add a Select operator with a DELTA strategy. Both strategies showed to be the most robust ones in the evaluations from Section 2.5.

## 7.2.3 Termination and Iteration

As described previously, after the rewrite stage the intermediate result is evaluated to finish the iteration. In Section 4.3 some features were presented that could be used for building a termination condition. However, the Monogamy feature shows to be most appropriate. It measures how close a result is to a 1:1 mapping. In principle, also other features with different tuning goals might be used instead. For instance, the matching system could try to ensure that all source elements find at least N partners. A quality criterion could then measure the overlap of top-N sets.

Typically, iterations run as follows. A first iteration identifies basic matches without considering structural aspects. Subsequent iterations propagate these matches to other element pairs based on structural information. This propagation is similar

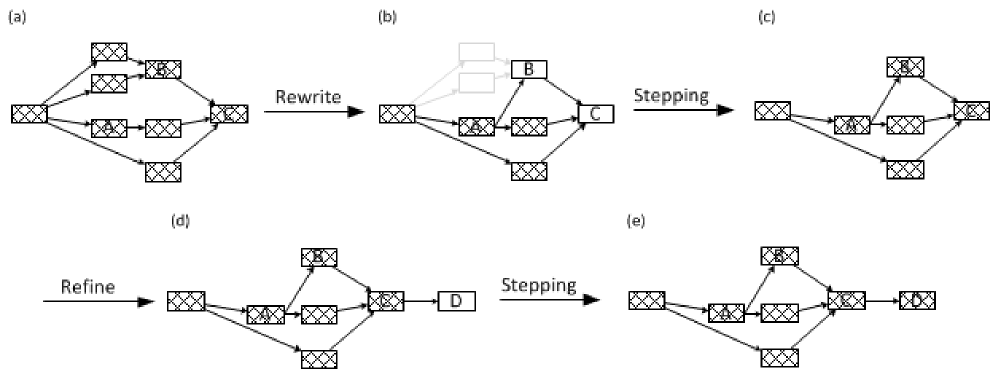


Figure 7.2: Stepwise execution of changed process parts

to what was intended with Similarity Flooding [117]. The difference is that no stable state is reached after multiple iterations and the direction of propagation is found adaptively. After each iteration, the Monogamy feature is computed. If the propagation reduces the computed Monogamy value by destroying basic matches then the process terminates. In cases of high structural similarity and limited seed matches from the basic matchers multiple iterations of Refine, Combine and Rewrite might be necessary.

### 7.2.4 Stepwise Matching Process Execution

A crucial element of the rule based execution is that rules may rely on intermediate results and their features to compute its relevance. For that reason every change of the matching process triggers the execution of newly added operators. If rewrite rules change the input to already executed operators then a re-execution of process parts might be necessary. In order to achieve re-execution of operators, the execution state of downstream operators is reset. When executing the process using the algorithm from Section 4.5.1, the already executed operators are skipped and operators with execution state NONE are executed. Figure 7.2 illustrates that behavior with a small example. A process (a) is given and all operators are executed as marked with the cross-pattern. The process is rewritten so that two operators are removed and a new edge is added to operator B (b). Operator B changed its input and needs to be recomputed as does C. After the step-wise execution (c) a refine rule is executed which adds operator D to the process (d). This time only the newly added operator needs to be executed.

## 7.3 Rule Collection

In the following section, specific rules and their rationale are described. The introduced rules are listed in Table 7.1. Some selected rules are presented in more detail

| Rule-Type | Rule |
|---|---|
| Refine (basic) | Add-Name |
| | Add-Complex-Type |
| | Add-Statistics |
| | Add-Annotation |
| | Add-Instance |
| | Add-Restriction |
| Refine | Add-Path |
| | Add-Children |
| | Add-Sibling |
| | Add-Leaf |

| Rule-Type | Rule |
|---|---|
| Rewrite | Noise-Filter |
| | Blocking-Filter |
| | Weight-Name |
| Combination | OWA-Most-Combine |
| | Average-Combine |
| Selection | Complex-Delta-Select |
| | Skimming |
| | Max1-Select |
| | Adaptive-Threshold |
| | Restrict-to-N:N |

Table 7.1: Rules introduced in this thesis

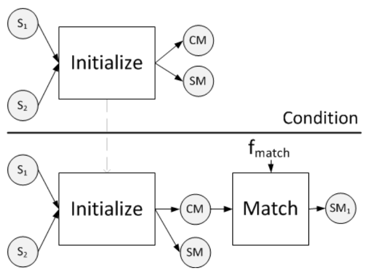


Figure 7.3: Refine rule pattern in the first iteration

whereas the others can be found in the Appendix B. The rules are grouped by the different stages that were presented above.

### 7.3.1 Refine Rules

Initially, refine rules are described that are used in the refine stage after startup of the adaptive process construction. In that stage, the rule selection solely depends on schema or schema similarity features since there are no intermediate similarity matrices that can be analyzed. Rules that trigger on startup have as input a matching process that only consists of Schema-Import operators and an Initialize operator with an empty output similarity matrix. The rule input and output patterns are illustrated within a rule template in Figure 7.3. The output pattern of most refine rules that trigger on startup from the Table 7.1 equal the one in Figure 7.3. A Match operator is added to the process that gets as input the comparison matrix CM. The operator does not rely on a constituent similarity matrix SM. However, also more complex process structures could be added within the first refine iteration as will be shown within the next paragraphs.

## Add-Name Rule

The Add-Name rule adds a name-based matcher to the process. The rule tries to formalize a heuristic which helps to decide when to use the token-based Name or the Edit-Distance matcher or none of them. The relevance condition of the Add-Name

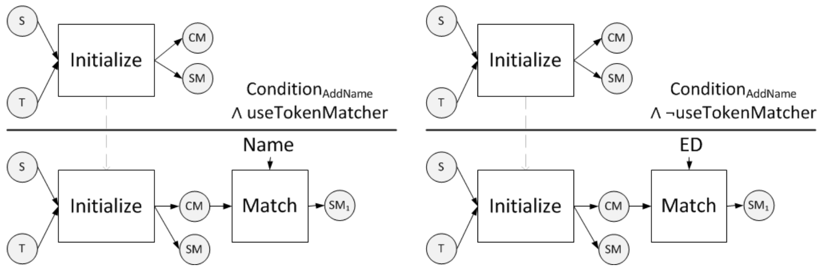


Figure 7.4: Add-Name rule

rule first assesses if there are names present in both the source and the target schema. If names are present which can be computed with the Name-Existence feature the information contained in the element names is evaluated. This can be done with the Name-Variance feature. If the Name-Variance is very low ($< 0.1$) in one of the input schemas then the information contained in the element names most probably does not help to compute trustful similarity values. All this can be formulated as a relevance condition:

$$\begin{aligned} Condition_{AddName} &= \neg(NameExistence(S) = 0 \lor NameExistence(T) = 0) \\ &\land(NameVariance(S) > 0.1 \land NameVariance(T) > 0.1) \end{aligned} \tag{7.1}$$

If the condition evaluates to true, a name-based matcher is added. The rule choses between the token-based Name matcher and the Edit-Distance matcher. For that purpose, the Element-Token-Ratio feature can be used. It assesses whether names consists of multiple tokens. An Element-Token-Ratio close to 1 implies that most element names only consist of one token. In such cases an Edit-Distance matcher can be more appropriate. Even if the token ratio is low the Name matcher might not perform well since some token overlap between the source and the target schema should be measureable. This can be done with the String-Token-Overlap feature. The resulting condition when to use the token-based matcher is shown below:

$$\begin{aligned} useTokenMatcher = Avg(ElementTokenRatio, S, T) &\leq 0.9 \\ \land StringTokenOverlap(S,T) &> 0.05 \end{aligned} \tag{7.2}$$

If the condition $Condition_{AddName}$ and $useTokenMatcher$ evaluate to true then a Match operator is added to the process parameterized with the Name matcher. If only the $useTokenMatcher$ condition is false, an Edit-Distance matcher will be used.

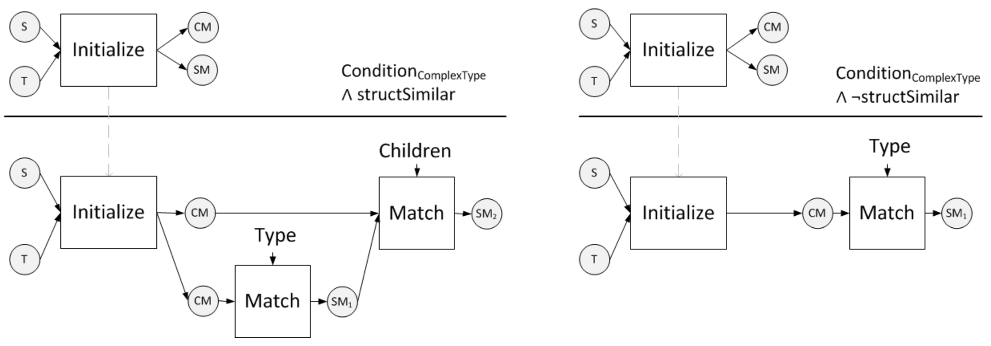


Figure 7.5: Add-Complex-Type rule

The thresholds for the individual features in the conditions are set tentatively from analyzing a small set of mapping scenarios. However, as will be discussed later, these values could also be automatically set using machine learning techniques.

## Add-Complex-Type Rule

When analyzing the behavior of the (Data-)Type matcher as it was proposed by [35] some observations could be made. The matcher is very restrictive in that it relies on a mapping table of known types. If two types are unknown and their names differ a similarity value 0 is returned. In particular XML-based schemas contain complex types that are constructed from a number of basic types. It can be observed that a combination of a Type with a Children matcher is able to identfy similarities between such complex types much better than a simple Type matcher.

Accordingly, the Add-Complex-Type rule adds a sub-process to the matching process. It consists of a Children matcher that takes the output similarity matrix of a Type matcher as constituent input. The rule is shown in Figure 7.5. For defining the relevance condition the Type-Existence feature is used. Moreover, the Type-Variance value should be high. A Type matcher can only return reasonable similarity values if the Type sets of the two schemas to be compared overlap. For that purpose, the schema similarity feature Type-Set-Overlap is used within the relevance condition:

$$\begin{aligned} Condition_{ComplexType} = \neg(&TypeExistence(S) \leq 0 \vee TypeExistence(T) \leq 0) \\ &\wedge TypeVariance(S) > 0.1 \wedge TypeVariance(T) > 0.1 \\ &\wedge TypeSetOverlap(S,T) > 0.3 \end{aligned} \tag{7.3}$$

Still, using the Children matcher requires that the two schemas are structurally similar. If they are not, which can be measured with the Structural-Similarity feature, a simple Type matcher is added.

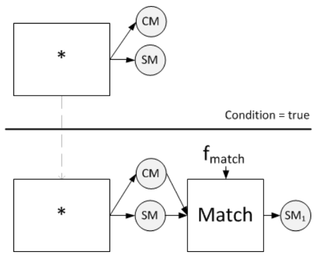


Figure 7.6: Refine rules

### Other Basic Refine Rules

Some matching systems from the ontology matching domain have matchers in their matching library that solely rely on structural statistics like the number of child and parent elements or the path length (see RiMOM or Falcon). For this thesis, a Parent- and Child-Count matcher was constructed that can be combined to a Statistics matcher. The Add-Statistics-Matcher rule adds such construct to the matching process (see Appendix Section B.1.1). The other refine rules that trigger in the first refine stage Add-Instance (Section B.1.3), Add-Annotation (Section B.1.2) and Add-Restriction (Section B.1.4) simply add a single matcher to the matching process as shown in the rule-template of Figure 7.3. They all primarily rely on the Attribute-existence features to formulate the relevance condition. The Add-Annotation rule additionally also measures the Attribute-Variance and String-Token-Overlap.

## 7.3.2 Refine Rules

After the first iteration of the staged execution, rules can trigger that depend on a constituent input matrix and computed matrix as well as matrix similarity features. Such rules mostly add structural matchers to the process as illustrated with the rule template from Figure 7.6. But also more complex structures could be added.

### Add-Path Rule

The Add-Path rule adds a Path matcher to the matching process (see Figure7.7).

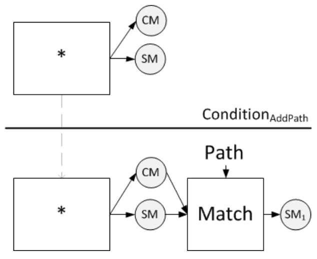


Figure 7.7: Add-Path rule

The Path matcher computes the similarity of two elements based on the constituent input similarity from the previous iteration. Typically, a Path matcher reduces the number of multi-mappings that may be created by basic matchers. This property was also exploited for large schemas in COMA++ with the Refine construct [11]. However, for smaller matching problems the Path matcher is always used in COMA++. Multi-mappings are often created when there are many repeating fragments in the source or target schema.

The relevance condition of the Add-Path rule relies on the Structural-Similarity feature that was described in Section 4.3.2. Since the Structural-Similarity values do not imply that the paths are similar (Structural-Similarity only measures similarities of node statistics) additional features such as Schema-Depth and Path-Variance are included to compute the relevance of the rule. From preliminary evaluations it could be observed that the Path-Variance seems to be a good indicator for using the Path matcher. In particular if the source and target structures differ the Path matcher could be helpful. Also if one of the schemas contains repeating fragments the probability of multi-mappings increases and therefore the Path matcher should be used. The relevance condition can then be formulated as follows:

$$\begin{aligned} Condition_{AddPath} = AVG(SchemaDepth, S, T) > 0.2 \\ \wedge FS(PathVariance, S, T) < 0.75 \\ \wedge FS(RepeatingFragments, S, T) < 0.95 \end{aligned} \tag{7.4}$$

### Add-Children Rule

The Add-Children rule adds a Children matcher to the process. Note, that even though the Children matcher might have already been added with the Add-Complex-Type rule, the two Children matchers have different input and therefore compute different results. Again, the Structural-Similarity feature is a good indicator for adding or not adding a Children matcher since it directly corresponds to the behavior of the matcher. However, child count statistics as measured by the Structural-Similarity feature are not sufficient to select or unselect the matcher. False mappings could be created due to switches in hierarchy levels so that a child in the source schema is a parent element in the target schema. To measure this, the Cross-Matches feature can be used. For each correspondence $c$ between elements $s$ and $t$ Cross-Matches checks whether child elements of $s$ also match to parent elements of the element $t$ or vice versa. Crossing matches are rare but if they appear the Children matcher should not be used. A second indicator that could be used is the Schema-Depth feature and its similarity in the source and target schema. If the difference between the values computed for the source and target is high, then the Children matcher should not be applied. The resulting condition is shown below:

$$\begin{aligned} Condition_{AddChildren} = StructuralSimilarity(S,T) > 0.25 \\ \wedge CrossMatches(SM) < 0.05 \\ \wedge FS(SchemaDepth, S, T) < 0.5 \end{aligned} \quad (7.5)$$

### Other Refine Rules

The Add-Sibling (see Section B.2.2) rule relies on a Sibling matcher specific feature, the Sibling-Distribution. The rationale is the following: If two matches point to siblings in the source they should also point to siblings in the target. The more such match pairs an input matrix contains, the more likely it is that applying a Sibling matcher is appropriate.

The Add-Leaf rule (see Section B.2.1) currently solely relies on the feature. A high feature value gives some indication whether to use the Leaf matcher or not. However, Leaf matcher specific features should be defined in future. For instance, a feature could take the input mapping from a basic matcher as indicator. If there are many non-leaf matches where the leafs of the source and target element also match then applying the Leaf matcher could be recommended.

## 7.3.3 Rewrite Rules

The set of rewrite rules is restricted to rules that optimize quality of the process. Rules for optimizing the performance are not included since all operators are already executed when the rules are applied and thus no additional performance improvements can be achieved. However, if the finally generated output process should be reused as a static process afterwards then such rules could be added to optimize the performance.

### Noise-Filter Rule

The Noise-Filter rule is concerned with so-called noise that can appear in a similarity matrix. It can be observed that combination strategies can have problems to cope with such noise. However, filtering noise should only be applied under certain conditions. In order to decide when to apply the Noise-Filter rule the Noise feature values of all inputs to the Combine operator need to be computed. If one of the inputs contains noise ($Condition_{NoiseFilter} = Noise(SM) > 0.3$), then the Noise-Filter rule is applied. The Noise-Filter rule adds a Select operator with a TRESHOLD selection strategy between the input operator that produces noise and the Combine operator as shown in Figure 7.8.

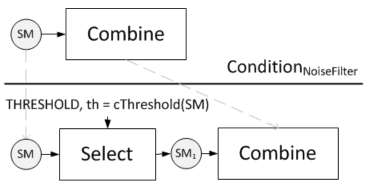


Figure 7.8: Noise-Filter rule

The selection threshold for the THRESHOLD selection is automatically computed with a function $cThreshold$ similar to the Noise feature. Details can be found in the Appendix Section B.3.1.

### Blocking-Filter Rule

The Blocking-Filter rule changes the order of operators to increase precision. Blocking is a well-known technique in the Object Matching area to reduce the number of pairwise computations [16]. As could be shown in [134] some precision improvements can be reached by filtering comparisons. The Blocking-Filter rule statically blocks computations of pairs by type. This is in line with observations from Giunchiglia et al. [66] where type similarities are used to correct computed mappings. The Blocking-Filter rule assumes that there was at least one Type matcher added to the process within a previous Refine stage. The Type matcher result is taken and all pairs below a given threshold are filtered. The goal is to execute all other operators only on the filtered set of comparisons. In order to prevent re-execution of operators None matchers are added that take the results of operators as constituent input and forward similarity values only for the pairs where the comparison matrix entry is true. The input pattern and applied changes are shown in the Blocking-Filter rule within Figure 7.9.

As with other rules, the Blocking-Filter rule should only be applied if certain relevance conditions hold. First, there should be a Type matcher in the current process. Secondly, it must be measured if the Blocking could have negative effects on recall. For that purpose a combination of all input similarity matrices $SM_1, \ldots, SM_N$ without the Type matcher result $SM_{dt}$ is computed. Then, the combined result is selected with a very restrictive set of selection strategies to compute a high precision result. If there are many pairs in the high precision matrix with similarity value $> 0$ that are low-valued in the Type matcher result (we refer to as countNegative), then the type blocking should not be done. This can be formulated as a relevance condition below:

$$Condition_{BlockingFilter} = \frac{countNegative}{Min(|S|, |T|)} > 0.01 \qquad (7.6)$$

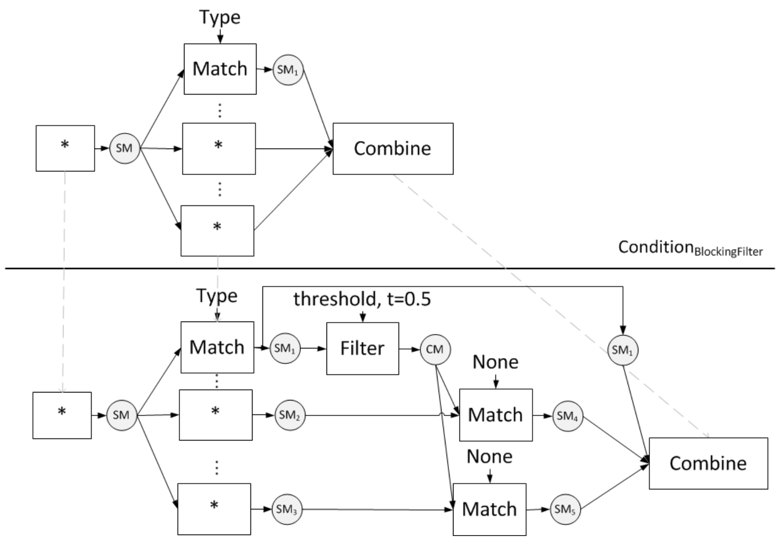


Figure 7.9: Blocking-Filter rule

### Weight-Name rule

Peukert et al. proposed in [138] to weight tokens in the Name matcher when the Element-Token-Ratio is low. Token weighting can help to cope with schema mapping problems where the set of words that are used to name schema elements is much smaller than the actual number of elements. This reuse of words leads to incorrect matches when using the common token-based Name matcher due to ambiguities introduced by frequent tokens. The Weight-Name rule simply replaces the Name matcher matching strategy with a NameWeighted matcher (see Figure 7.10). The

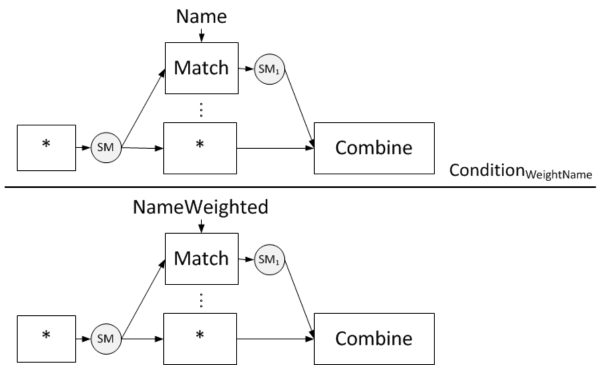


Figure 7.10: Weight-Name rule

relevance of the rule can be computed from the Element-Token-Ratio, RepeatingFragments and the TopN-Overlap.

$$\begin{aligned} Condition_{WeightName} = AVG(RepeatingFragments, S, T) > 0.85 \\ \wedge(ElementTokenRatio(S) < 0.7 \vee ElementTokenRatio(T) < 0.7) \\ \wedge TopNOverlap(1, SM_1) < 0.5 \end{aligned} \tag{7.7}$$

Note that the decision to use the NameWeighted matcher also relies on a matrix feature which could not be computed within the starting rule that adds the name-based Matcher.

### 7.3.4 Combination Rules

In Chapter 2 a number of different combination and selection approaches were presented. The combination rules that are described below only add a single Combine operator to a process. They differ in the chosen combination strategy and the relevance condition. However, also more complex combination approaches are feasible that consist of multiple combinations.

#### OWA-Most and Average-Rule

The main observation in the evaluations of combination strategies from Section 2.5 was, that in most cases the AVERAGE combination returned best results. Only in some cases the OWA strategy was able to compete. The problem is to choose the most appropriate strategy for a given mapping problem. Surprisingly, also the authors of OWA [88] discuss that the OWA strategy cannot be the same for all possible mapping problems and used matchers. They describe that the most appropriate strategy could be chosen by observation instead of training. The user should take several schema element pairs and evaluate the similarity values that are returned by the matchers to be combined. Based on how many of these matchers return a similarity value greater zero they propose to select the most appropriate OWA operator. If almost every similarity measure computes a value bigger 0 the OWA Most operator should be used. The Commonality feature from Section 4.3.4 automatically measures that. It computes how big the agreement between similarity matrix inputs to a Combine operator is. From experimenting with different values of the Commonality feature an appropriate relevance condition for the OWA-Most-Combine rule could be derived:

$$Condition_{OWAMost} = Commonality(SM_1 \ldots, SM_N) > 0.4 \tag{7.8}$$

In all other cases the Average-Combine rule is used by default.

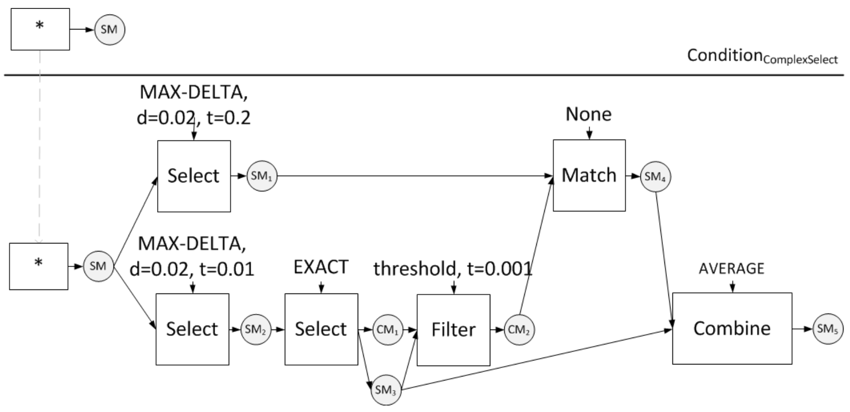


Figure 7.11: Complex-Delta-Select rule

### 7.3.5 Selection Rules

After the final iteration of the stage-wise execution, selection rules can be executed that append Select operators to the matching process. Some of the listed selection rules from Table 7.1 are described in detail whereas others can be found in the Appendix B.4.

#### Complex-Delta-Select Rule

In Section 2.5 also selection strategies were analyzed. The most robust selection strategy was the MAX-DELTA strategy. The Complex-Delta-Select rule adds a Select operator with such a selection strategy to the matching process. However, it extends that selection with a complex subprocess that consists of further select operators and a Filter operator that help to increase recall and precision. The complete rule is shown in Figure 7.11.

The rationale is the following: The input result is selected with a very low threshold close to 0. The following EXACT selection (see Section 2.4.2) restricts the result to the 1:1 matches. As was discussed in Section 2.4, the 1:1 matches often have a higher precision. For all other pairs that do not involve 1:1 matches the MAX-DELTA selection result with higher threshold is computed. The Filter operator together with the None matcher allows extracting the other pairs. Finally, the 1:1 result and the MAX-DELTA result is combined. Currently, the Complex-Delta-Select rule always triggers so that the relevance condition $Condition_{ComplexSelect}$ is true. However, in future this could be changed by measuring the probability that a mapping problem requires a 1:1 mapping result such as size differences or the amount of n:m correspondences.

### Skimming Rule

In Section 4.4 the skimming pattern was described that takes a union of the most probable correspondences from multiple iterations of selection and match. With the Skimming rule this behavior is implemented as a selection rule. The input to the selection is the most recently selected node and all result mappings of previous iterations that ended with a Combine operator. The most probable results of each of these Combine operators are skimmed with a high precision selection. The skimming results and the selection result of the most recent selection are then combined using a Combine operator (see Figure 7.12).

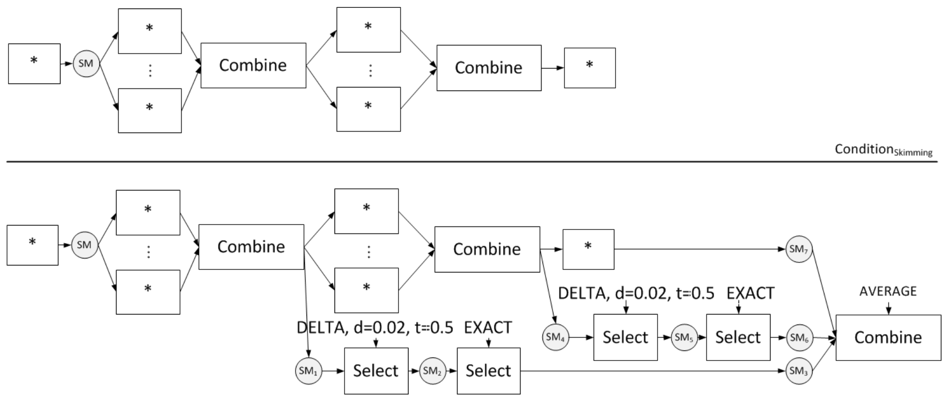


Figure 7.12: Skimming-Select rule

The skimming is particularly helpful if the intersection of mapping results from later iterations differ strongly with mapping results from early iterations. this can happen if basic matches are propagated by structural matchers to other element pairs and the original element pairs do not have high structural similarity. Without skimming the potential correspondence candidates could get lost. Such differences can be identified with the Commonality feature. The resulting relevance condition is defined as follows:

$$Condition_{Skimming} = Commonality() > 0.3 \tag{7.9}$$

### Further Selection Rules

In many mapping scenarios only the top-1 result for a selection is needed. In particular if schema elements and fragments do not repeat, the top-1 result should contain the correct match. The Max1-Select rule (see Appendix Section B.4.1) adds a single Select operator with a MAX-N selection strategy to the process. It can be observed that for many mapping problems it is possible to measure with the Repeating-Elements feature what selection strategy should be chosen. If elements are not repeating in the source and target schema then a MAX-N selection with N=1 is could be beneficial.

The Adaptive-Threshold rule (Appendix Section B.4.2) adds a Select operator with THRESHOLD strategy to the most recent selection. A problem in many matching systems is the definition of appropriate thresholds. The Monogamy feature value is used to compute such a selection threshold from the input mapping by testing different thresholds with a default MAX-DELTA selection. The threshold that results in the highest Monogamy value is then taken as threshold for the THRESHOLD strategy.

The *Restrict-to-N:N* rule (Appendix Section B.4.3) restricts the cardinality of a mapping. In particular if schemas contain repeating substructures it is more probable that the correct match results contain multi-mappings that need to be restricted. It could also add a Select operator with the EXACT selection strategy. This is relevant if the mapping tends to be a 1:1 mapping and there are still some multi-matches included in a matrix.

## 7.4 Adaptive Execution Examples

For illustrating the interplay between the staged execution, features and rules three detailed examples of adaptive process execution are given. To simplify the presentation all selection rules except the Max1-Select and the Adaptive-Threshold rule are left out when running through the examples. The first example relies on an artificial simple mapping problem between two order schemas. The second example also relies on the artificial problem with slightly changed schemas. The third example uses the ANATOMY mapping problem from the OAEI evaluations.

### 7.4.1 Simple Order Example

Two given schemas represent orders from different companies. They are trees that contain element names, types and annotations. The schemas and the correct mapping are visualized in Figure 7.13.

The source order schema is structured into a header part (Hdr) and a contact part (Cnt). Element names of the source are short and cryptic. The annotations of the source schema elements are meaningful. The target schema is similarly structured. The element names are longer and meaningful. The annotations of the target elements are machine generated text that does not give any value for matching.

The adaptive process starts with the Initialization stage where an Initialize operator is added to the process. Afterwards, the first iteration of Refine, Rewrite and Combine starts. In the Refine stage only the Add-Complex-Type and the Add-Name rule trigger (see matching process in Figure 7.14a). The Type and Children matcher are added since the StructuralSimilarity between the source and the target schema is high ($= 0.9$). Since all elements in the source and target schema contain names, the Name-Existence value is $1$. For name matching the Edit-Distance matcher (ed) is selected since the computed Element-Token-Ratio is 1 for both schemas which means that there are no tokens reused in the schemas. Moreover no overlap of name tokens can be measured. Even though the Annotation-Existence value is high for both

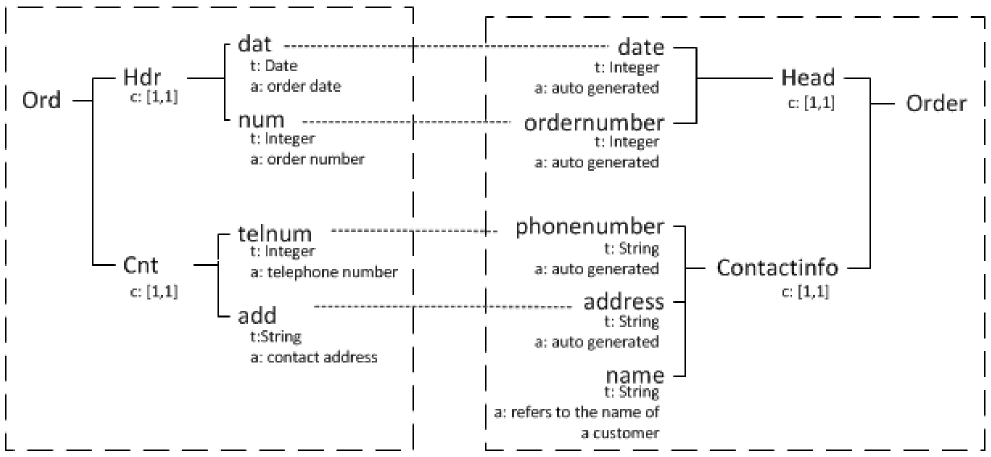


Figure 7.13: Mapping example

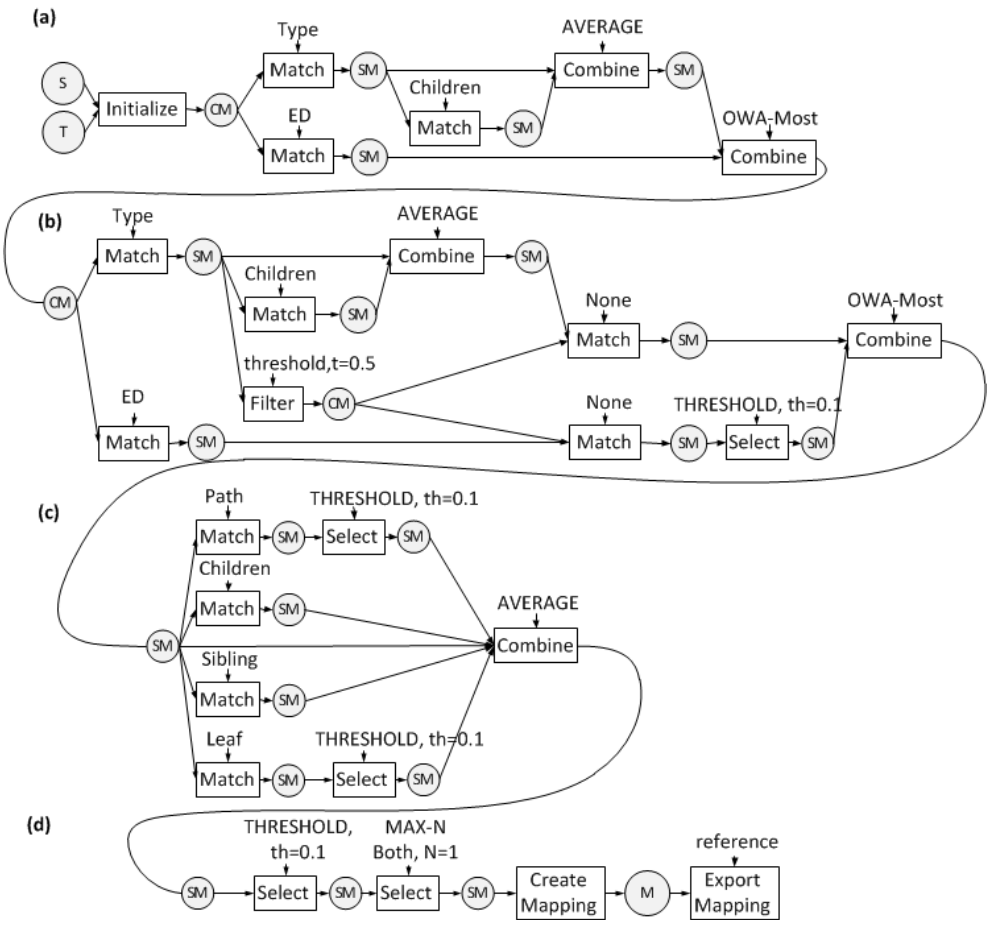


Figure 7.14: Example: Adaptive construction of a matching process

schemas a low Annotation-Variance (0.31) in the target schema is measured and therefore the relevance condition of the Add-Annotation rule computes to false. Since there are no instances, cardinalities or restrictions available, the Add-Instance- and Add-Restriction rule do not trigger. As combination strategy the OWA-Most strategy is chosen since the computed Commonality of both matchers from the refine stage is 0.66 which is bigger than 0.4 as required by the relevance condition of the OWA-Most-Combine rule. As rewrite rules, the Blocking-Filter rule and the Noise-Filter rule are executed. The Weight-Name rule does not trigger since no token-based Name matcher exists and there are no repeating tokens in the element names of both input schemas. The Blocking-Filter rule is applied since the comparison of the Type matcher result and a high precision selected Name matcher result did not reveal any false negatives. Since both matchers were already executed the results of the Edit-Distance matcher and the Type matcher are filtered by using the None matcher. It forwards similarity values of the constituent mapping if a comparison matrix entry is true. The Noise-Filter adds a Select operator after the None matcher that results from filtering the Edit-Distance matcher. The rewritten process is shown in Figure 7.14b. After that a selection is performed on the output and the Monogamy value is computed which is 0.85. This is due to ambiguity of the “telnum” and “num” element in the source schema that match to “phonenumber” and “ordernumber” in the target.
Figure 7.14c shows the matching process after the next iteration. Due to high Structural-Similarity of the two input schemas 4 structural matchers where added. The Noise-Filter rewrite again added Selection operators for the Path matcher and the Leaf matcher result due to high computed noise. As combination strategy AVERAGE is selected when adding a Combine operator. Again a selection is performed and the Monogamy value is computed. This time, the Monogamy is 1 so that a further iteration can be started. In that iteration again a number of structural matchers are added. The computed Monogamy value of that iteration also computes to 1. Since no improvement could be measured the iteration is reverted and a final selection phase is started.

The Adaptive-Threshold and the Max-1 Select rule trigger and add Select operators (see Figure 7.14d). The Max1-Select-rule is triggered since the schemas do not contain repeating elements and the intermediate result does not consist of multi-mappings. The process is finished by adding a Create-Mapping operator and the Export-Mapping operator. The finally achieved FMeasure is 1 since all matches were identified correcly.

### 7.4.2 Simple Order Example - Changed Structure

The second small mapping problem is constructed by changing the structure of the target schema. The target schema now only contains a single root element with all other elements grouped below as shown in Figure 7.15.

The adaptive process again starts with adding an Initialize operator. Then, the Type and Edit-Distance matchers are added. This time, only the simple Type matcher

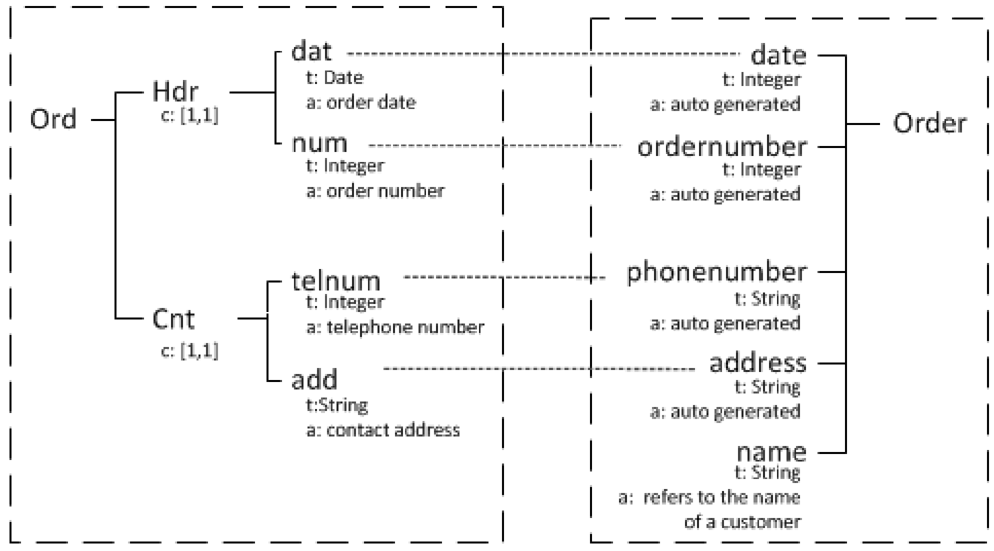


Figure 7.15: Example mapping problem with low structural similarity

is added since the Structural-Similarity is low ($0.25$). As combination rule OWA-Most-Combine is chosen and the Blocking-Filter rule as well as the Noise-Filter rule are executed similar to the example above. The resulting process after the first iteration is shown in Figure 7.16b. The computed Monogamy value after the iteration is $0.66$.

In the following refinement stage, only a Path matcher operator is added (see Add-Path rule) since the average Schema-Depth is greater than $0.2$ and the difference of Path-Variance is smaller than 0.75. The Noise-Filter rule triggers, a Combine operator with AVERAGE strategy is added. The computed Monogamy value decreased after the iteration so that its result is reverted. Finally, the Selection stage starts and adds similar Select operators as before. The process was not able to correcly match telnum to $phonenumber$ and $num$ to $ordernumber$ with the available meta-data which resulted in a lower f-measure. However, a synonym-based matcher could be used to improve that.

### 7.4.3 Example 3 - Mouse-Anatomy to NCI Thesaurus

The Mouse Anatomy to NCI-Thesaurus ontology mapping is part of the OAEI evaluation. When executing the adaptive process with the ANATOMY mapping problem the following process is performed. As with the previous example a Type and a Name matcher is selected in the first Refine stage (see Figure 7.17a). In contrast to the previous example a Name matcher is selected since the reuse of name tokens is high within the ANATOMY schemas ($ElementTokenRatio(S) = 0.59$, $ElementTokenRatio(T) = 0.66$). In the Rewrite stage the Blocking-Filter and the Noise-Filter rule are executed to filter the Name matcher result. The computed

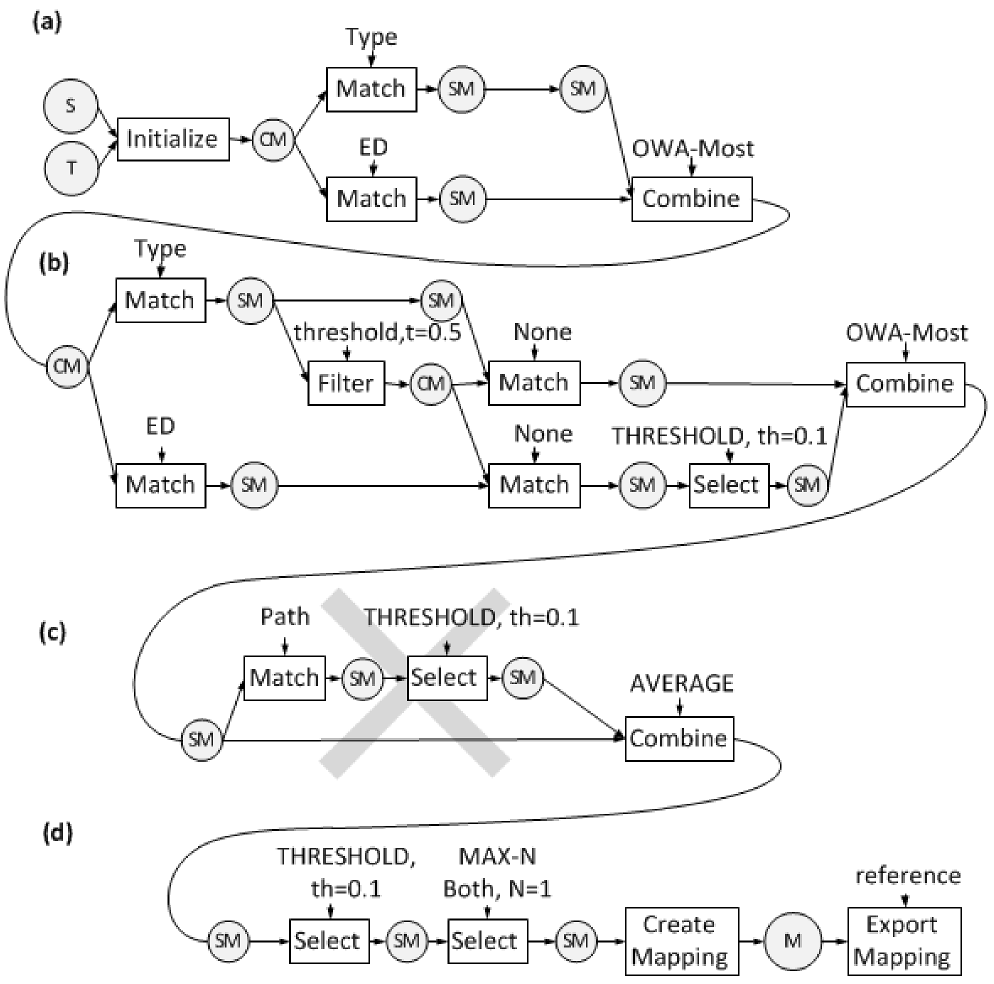


Figure 7.16: Stagewise construction of process for example 2

Monogamy value after the first iteration is $0.52$. In the Refine stage of the following iteration only the Add-Sibling rule triggers. This is due to low a Structural-Similarity ($StructuralSimilarity = 0.34$) and a high Repeating-Fragments value ($RepeatingFragments = 1$). Also the resulting mapping from the first iteration does contain a number of crossing matches which effects that the Children matcher is not added. The Add-Sibling rule triggers since the Structural-Similarity is greater than $0.3$ and the Sibling-Distribution feature value of the mapping computed in the first iteration is lower than $0.3$. The computed Monogamy value at the end of the iteration slightly decreased to 0.51 so that the iteration is reverted and the Selection stage is started. A small Multi-Mappings value and high Repeating-Fragments value leads to choosing the MAX-1 selection strategy. The achieved F-Measure is 0.83 which is lower that the achieved F-Measure from AgreementMaker in the OAEI contests but similar to the result Falcon achieved.

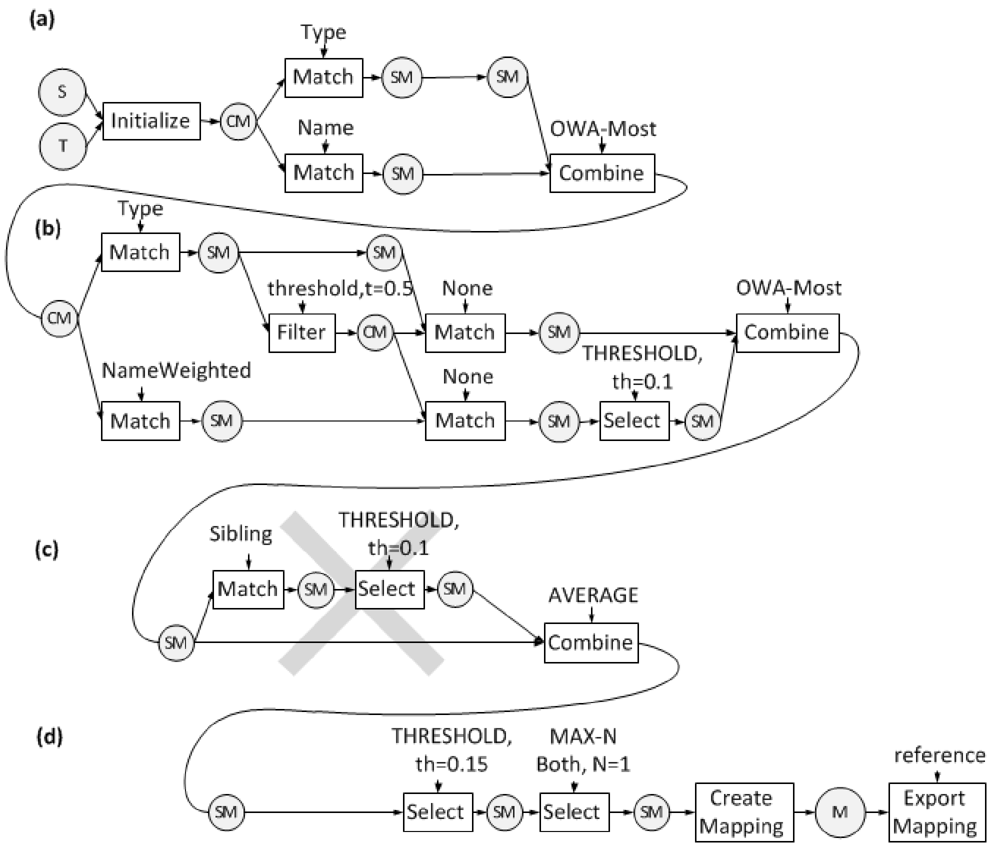


Figure 7.17: Stagewise construction of process for example 3

## 7.5 Adaptive Matching System

An adaptive matching system was built that incorporates the above presented concepts. It consists of an adaptive process construction component and a process execution engine similar to the one presented in Section 4.5 (see Figure 7.18). To solve a mapping problem, the matching system obtains two schemas as input and returns a mapping as output. Ideally, no further parameterization input should be needed. All necessary parameters should be defined automatically. The system consists of a registry that contains a number of features, matching rules as well as a component library that contains all necessary operators and strategies in particular the matchers, combination or selection strategies. The core component of the system is the adaptive process construction that basically implements the proposed staged rule application approach. In a preprocessing step all schema features of the input schemas are computed and cached to avoid double computation. After every change of the process the matching process execution is called to execute the new operators. This creates new intermediate similarity matrices that can be analyzed with subsequent matrix features. Currently, the adaptive execution always starts with an

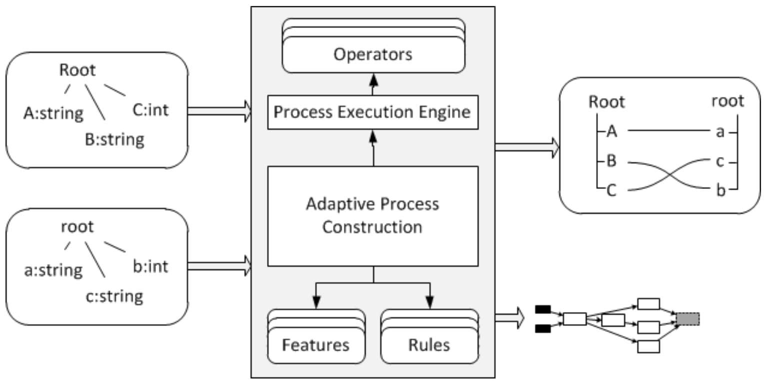


Figure 7.18: Adaptive matching system architecture

empty process and no input mapping is provided. However, it is possible to extend the approach so that an existing mapping can be providd which is then refined by the adaptive process construction.

Rules are implemented as plain JAVA-classes. It was also considered to use a graph rewrite system like AGG [160] to perform rewrites. Such system could be used since a rule execution is very similar to in-place graph transformations [33]. In particular matching the input pattern with the current process and finding instances of the pattern is well supported. However, the additional conditions that need to be executed onto found pattern instances imposed bigger changes to the execution of the AGG-engine. Since in the current system the rule set is restricted and the rule execution time is not an issue, a plain JAVA implementation of rules is sufficient. The integration of an in-place graph transformation engine is left open for future work.

## 7.6 Discussing Design Decisions

In the beginning of the chapter, it was argued that a single set of rewrite rules would compete if treated equal. For that purpose a staged execution of rules with special rule types was introduced. The structural diversity an adaptively created process can then have is limited. This simplifies the rule selection process significantly. The restrictions made it possible to build a highly adaptive schema matching system while ensuring control and termination. Also problems with the order of rule application are tackled.

Another question that could be discussed is why manual tuning of rules should be better than tuning an existing matching system and its parameters. The assumption that is made is, that rules are more stable in their parameters and should not need tuning for new mapping problems. Most of the presented rules do encode heuristics that are valid in different domains of mapping problems.

A further point to discuss is the dependency on the Monogamy feature for finishing iterations in the staged process or for finding thresholds in the selection phase. This favors matching processes that produce 1:1 mappings. In principle other quality measures and matching goals of the overall system would be feasible as already shortly pointed out above. For instance, instead of returning 1:1 mappings, the system could return the top-N matching candidates for each element of a source or target schema. The quality measure could be changed to a feature that measures how close a similarity matrix is to the intended top-N mapping. However, also changes to the refine, combination and selection rules would be necessary so that they favor selection and combination strategies that optimize towards a topN result as was proposed with the overlap reduction by Peukert et al. [138]. For instance, an overlap reducing combination would be feasible.

Furthermore the execution time can be an issue. Schema features are computed upfront. That could easily be changed to on-demand computation. It only needs to be ensured that they are not computed multiple times. Operators are not executed twice unless their input changed and required additional execution. Also reverting effects of a whole iteration is not helpful. However, executing and analyzing the intermediate results of operators is crucial for automatically choosing additional rules and computing their relevance. Note that there is a tradeoff in how to define relevance conditions. A relevance condition can be defined in a way that an operator or pattern is added to the process in most cases. Only in cases where a quality decrease is expected the rule computes to not being relevant. This could work, since combination and selection strategies are relatively robust to additional inputs. Still, it results in more operator executions and decreased performance. On the other hand the performance could be increased significantly if rules are less probably executed, by being optimistic. The drawback is that it may lead to missing a matcher that relies on a certain schema attribute and thus it may lead to reduced quality. In general, a good rule should consist of a relevance condition that correlates well with the quality improvement a rule could achieve.

Another approach to increase performance is restricting the number of rules that can be applied in a stage. Too many rules could hurt performance. However, the relative importance of rules cannot be computed easily. Thus, the user needs to define prioritized lists of rules to be applied. To increase performance, the list of rules can then be cut. For big lists of rules the cutting should also be done before computing the relevancies to reduce effort.

In Chapter 6 rewrite rules were introduced for increasing performance of a given matching process. This concept is not directly compatible with the adaptive rule selection approach. The reason is the following: In the adaptive system the choice of rewrite rules relies on intermediate matching results. Therefore, in the Rewrite stage all matchers are already executed once. Hence, changing the order and filtering comparisons cannot increase performance any more. Since after the rewrite phase no selection operator is present a selection push-down as advertised is not possible. Only the Blocking-Filter rule can be classified as performance increasing rewrite rule. Still, if the output process of the adaptive system should be reused, then a post-processing step could perform performance-oriented rewrites. Alternatively for large schemas, a user could be asked to select sub-schemas for which a process is constructed adaptively. The created process could then be rewritten to be performance-optimized and executed on the complete schemas.

## 7.7 Improvement over Related Work

The presented rule-based approach to adaptively create a matching process while execution goes beyond the current state of the art in a number of areas. First, features in the newly presented system are treated as first class citizens which makes a major difference to some presented automatic configuration approaches that were presented in Section 3.2. The eTuner system [98] only relies on a synthetically generated reference-mapping to tune a given fixed process without considering features of the schemas. Eckert et al. [49] were the first to recognize that features are crucial for improving the robustness of a matching process. Still, only schema and schema similarity features were included in a learning process of decision trees which according to the authors tend to overfit. Recently, Cruz et al. [30] presented an extension of AgreementMaker that also relies on schema and schema similarity features to learn the selection of the most appropriate matching process from a set of given processes. Like all learning-based approaches they still heavily rely on given reference mappings for training which is often not present for new mapping problems. In the approach presented in this thesis no reference mappings are needed for training. Moreover, also matrix and matrix similarity features are used to define rules.

The insight that rule-based selection of whole matching processes [121, 56, 139, 30] or selecting parts of existing matching process [99, 77] could help to build robust matching systems is not new. Some years ago the RiMOM and Falcon system already contained two fixed conditions for selecting and unselecting certain matchers within their fixed matching processes. Within the approach presented in this thesis such conditions can now be represented as generic refine rules. Mochol and Jentzsch [120, 56] focused on recommending the use of complete matching systems which is similar to the approach from Cruz et al. that tries to recommend a complete matching process in the AgreementMaker system. Still, the approach presented in this thesis is more ambitious. It tries to further disassemble the individual building blocks of strong matching processes as rules. It makes a feature-based selection of these rules while constructing a complete matching process. The only system that follows that path is the UFOMe system [139]. They also rely on fine-grained rules. However, their workflow seems to be fixed and restricted to a small set of rules that adapt the rules presented with RiMOM and Falcon. Used rules only rely on schema and schema similarity features. Due to the limited number of rules there was no investigation of how to represent rules, when to trigger rules and in what order to execute them. This thesis goes beyond that work in a number of areas. First, rules rely on matrix and matrix similarity features or intermediate results in addition to schema and schema similarity features. A staged rule execution approach is presented that describes when to choose what type of rule and in what order to cope with possible conflicts. Rules are presented in a semi-formal way that allows for reuse. This has some commonalities with defined rules that were presented by Ryu et al. [148] for selecting similarity measures for similarity search.

Still, the presented approach could be extended in a number of areas. Since the thesis pioneers the representation of matching process building-blocks as rewrite rules, further rules can certainly be found and can be integrated in the adaptive process. As described above, rule execution can be supported by a graph transformation engine to simplify the implementation of rules containing complex patterns and actions. The presented approach is orthogonal to learning approaches in that the parameters of rules could be learned from training data to further adapt the rules to specific domains. First experiments were made within the diploma thesis of Julian Eberius [48] that was done in the context of this doctoral work to learn parameters of complex rewrite rules by using the weka library.

# Part IV

# Evaluation

Chapter 8

# Evaluation

This chapter evaluates the rewrite-based approach for performance optimization and for adaptive matching process construction. Initially, the rule-based performance-optimization approach is evaluated by considering the following aspects:

- Can significant reductions of the overall execution time be achieved by rewriting a matching process?
- In comparison to that, what can be achieved with preprocessing for a matching process with name-based matchers?
- What is the influence of schema size on the achievable optimization?
- What can be achieved with the dynamic Filter operator if the final selection uses a THRESHOLD or a MAX-DELTA strategy?
- How does the number of matchers and the type of selection influence the performance optimization potential?
- What is the optimal setting of the threshold when using a threshold-based Filter operator and how does that influence result quality?

In the second part of the evaluation the adaptive matching technique from Chapter 7 is evaluated. In particular the effectiveness and robustness of the approach is analyzed. For that reason the effectiveness of the adaptive schema matching system is compared to currently known alternative approaches. Moreover, the behavior of individual rewrite rules is assessed and the diversity of used matching configurations is analyzed. The following questions drive the evaluation of the adaptive schema matching approach:

- Is the proposed rule-based matching approach robust, so that it returns good results for very different mapping problems?
- How do individual rules contribute to the overall quality of the adaptive matching system?

- How diverse are automatically computed matching processes? If only one type of matching process is constructed the need for adaptivity could be questioned.
- Does the Monogamy feature help to terminate iterations of the rule selection and execution process?

All evaluations are performed with a large set of diverse schema mapping problems for which reference mappings exist. These reference mappings are initially described in detail. Then, each evaluation part describes a setup and evaluation results. Finally, all results are summarized and the aforementioned evaluation aspects are discussed.

## 8.1 Data Set

The following evaluations rely on five different sets of mapping problems that are listed in Table 8.1. These sets contain a wide range of problems containing large- and small-sized schemas, schemas with cryptic element names, some with instances and some with flat or deeply nested structures.
To be comparable to existing work some publicly available mapping data sets are used in the evaluation. For instance, the Purchase Order (PO) data set consists of 5 schemas and 10 mapping problems. It was used in the early evaluations of COMA [35]. The schemas of the PO set are small- to medium-sized. Element names are human-readable and often consist of multiple tokens. The schemas do not contain annotations and instances. The OAEI 2010 Benchmark [53] consists of 110 synthetic mapping problems. The Benchmark mappings were created by perturbing existing schemas of a single mapping between two small ontologies. Names are removed or replaced by random strings, structure is flattened and instances are removed. Even though the data set offers a huge variety of mapping problems the actual mappings are rather artificial. Many systems that take part in the OAEI contest seem to tune specifically towards this data-set with design decisions that might not work for real world mapping problems. This will be discussed within the subsequent evaluation. The OAEI contest also offers real world mapping problems of large scale. From those problems the ANATOMY mapping problem is used in the evaluation. The mapping scenario is large-sized with high schema similarity. That means that about 60 percent of correspondences are trivial due to a very high name similarity. Instances are not contained. The Enterprise Services (ES) data set was created from extracting mappings from an SAP PI System [169]. It contains a variety of small to big mappings between service interfaces that are contained in the SAP Enterprise Services Repository. The schemas are heterogeneous and stem from different type systems such as SAP Intermediate Document Format (IDOC), Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT) or XSD. In particular within IDOC schemas, names are often cryptic. Most of the schemas contain annotations that can be used for matching. The EDIFACT set consists of a number of mappings that were created manually for an evaluation of the so-called Warp10 mapping repository within SAP [151].

| Mapping Data Set | # of Mapping Tasks | Square-root of the Cross-Product Size (STsize) min/average/max | # of Correspondences per Task min/average/max | Description |
|---|---|---|---|---|
| OAEI Benchmark (OAEI) | 110 | 53/84/112 | 29/71/97 | OWL, perturbed names & structure |
| Purchase Orders (PO) | 10 | 46/108/72 | 36/56/85 | XSD, readable names |
| Enterprise Services (ES) | 48 | 5/137/1307 | 4/119/1307 | XSD, IDOC, Edifact, cryptic names |
| Edifact (EDI) | 6 | 62/152/293 | 4/29/81 | Edifact/IDOC, cryptic names |
| Mouse - NCI Thesaurus (ANA) | 1 | 3005 | 1516 | Taxonomies, readable names |

Table 8.1: Schema matching evaluation data set

Table 8.1 lists the considered mapping sets together with some statistics. The size of the mapping problems $STsize$ is represented by the square root of the cross product size:$STsize = \sqrt{|S| * |T|}$. For each set, the minimum, average and maximum $STsize$ is shown. Also the minimum, average and maximum numbers of correspondences within mapping problems of a set are added. In addition, a short textual description of each data set it given. For example the ES data set consists of 48 mappings. The average size of the square-root of the cross-product size is 137. However also some mappings with $STsize$ up to 1307 are contained. On average each mapping consists of 119 correspondences. Again, a number of larger mapping problems do contain more correspondences. In the Appendix C a complete list of mapping problems together with further statistics and computed schema feature values are given.

For the performance evaluation all mapping cases are put into four groups:

- SMALL: The size of the cross product of small mapping problems is smaller than 1000.
- MID: Mappings that require less than 10000 comparisons are mid-sized mappings.
- LARGE: Mapping that require up to 100000 comparisons are classified as large mappings.
- XLARGE: All mapping cases with a cross-product size greater 100000 are classified as extra-large.

The subsequent performance evaluation is done per group in order to assess what influence the schema size has on the ability to improve the performance by pruning comparisons.

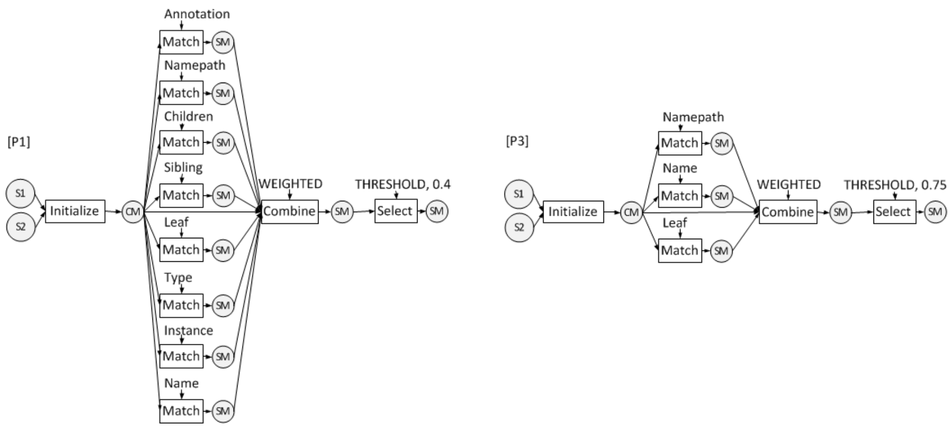


Figure 8.1: Parallel matching processes

## 8.2 Evaluating Rewrite-based Performance Optimization

Three parallel matching processes P1, P2 and P3 where created (see Figure 8.1 for P1 and P3). P1 consists of 8 matchers (Name, Namepath, Leaf, Children, Annotation, Type, Instance, Sibling). Their results are combined with a WEIGHTED combination. Each matcher gets equal weights assigned. The result is selected using a THRESHOLD strategy with a rather low threshold of 0.4. This low threshold is necessary since a number of the mapping problems above do not have any instances or annotations so that the respective matchers contribute with zero similarity to the combination which leads to mostly smaller similarity values. P2 is similar to P1 but an additional MAX-DELTA selection is added that returns better precision with slightly decreasing recall. MAX-DELTA is commonly used in practice. With P3 a matching process is created that only consists of three matchers (Name, Namepath, Leaf) and a threshold-based selection to show possible performance improvements with only few matchers.

The incidence graph from Section 6.2.1 is computed on a subset of 10 schemas from the ES data set for a matching library of 7 matchers which consists of the Name, Namepath, Leaf, Child, Sibling, Type and Annotation matcher (see Appendix C.2). Then, three different rewrites are executed and new matching processes are generated accordingly. First, the filter-based rewrite rule $Rule_{threshold}$ is applied onto each process (P1, P2, P3) which adds threshold-based static Filter operators for filtering intermediate results. The created processes are called SEQ_TH(P1), SEQ_TH(P2) and SEQ_TH(P3). Secondly, the same rule with a dynamic Filter operator is executed which creates processes SEQ_DYN(P1), SEQ_DYN(P2) and SEQ_DYN(P3). All processes SEQ_DYN and SEQ_TH still contain parallel parts. With the third test the remaining parallel parts within the SEQ_TH processes are sequentialized with a dynamic filter rule. The resulting processes SEQ_DYN_TH(P1), SEQ_DYN_TH(P2) and SEQ_DYN_TH(P3) do not contain parallel parts except from matchers that are not part of the incidence graph. The rewritten matching processes for P3 are shown

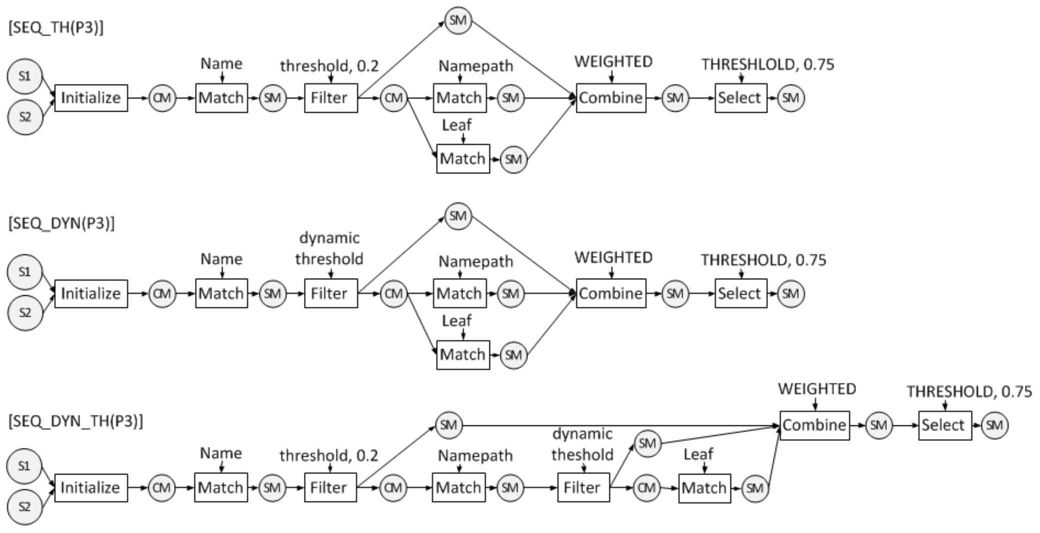


Figure 8.2: Rewritten matching processes for S3

in Figure 8.2. The rewritten processes of P1 and P2 can be found in the Appendix C.3. In order to be able to compare to a performance optimization technique that is not process-based a preprocessing step was implemented that precomputes token similarities for name-based matchers. By computing these token similarities once, all name-based matching can be accelerated since only existing values need to be combined. All experiments are run with preprocessing and without preprocessing.

The rewritten matching processes are executed on the given data sets from above. The execution time is measured in seconds. For quantifying the quality the common precision, recall and f-measure are used. Precision represents the ratio of correct correspondences among all found correspondences. Recall measures the ratio of all correct correspondences to the number of intended correspondences. F-measure then computes the harmonic mean of recall and precision.

### 8.2.1 Performance Comparison Results

Figure 8.3 shows execution times of P1 and its rewritten processes. Separate charts are provided for each group of schemas that are SMALL, MID, LARGE and XLARGE. For each group the average execution time without token preprocessing and with preprocessing was measured. Note that the computed f-measure for each group did not deteriorate, and only in some rare cases some comparisons were wrongly filtered. This was achieved by using a conservative filter threshold to not prune element pairs that might be relevant for the final result. Except for the dynamic rewrite rule the execution times of the rewritten processes are significantly smaller. The performance improves by a factor 4 even with increasing problem size. What can clearly be observed is the strong effect that the precomputation of token similarities has. This effect gets stronger with increasing schema sizes. However, the relative speedup that

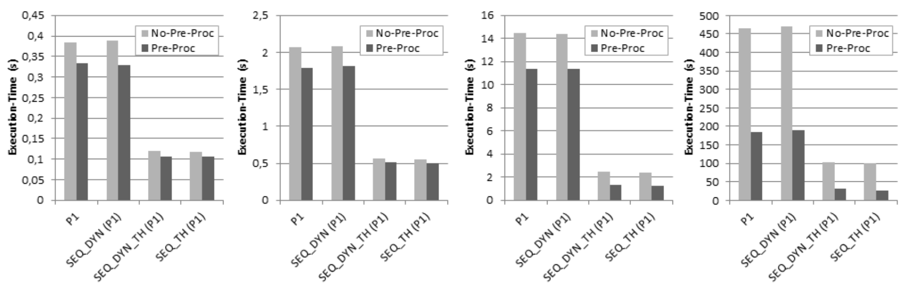


Figure 8.3: Performance evaluation of P1 and its rewritten processes

can be achieved with rewriting the process is stronger. What can be highlighted is that both performance optimization techniques can be combined to achieve major reductions in the execution time. Still, the missing effect of the dynamic rewrite rule needs to be discussed since a former evaluation showed some effect with a smaller matching process [134]. In these evaluations a 30% increase could be achieved with dynamic filtering of a matching process with a THESHOLD selection. The reason that this effect could not be reproduced is the small threshold of 0.4 and the higher number of matchers. For not yet executed matchers, the dynamic filtering assumes computed similarity values of 1. This means that more than half of all matchers need to be executed until first comparisons could be pruned. In the example, this almost never happens. In the experiments with the matching process P3 that has a higher threshold and less matchers the dynamic filtering achieves the intended effect.

In the initial publication of the work [134] dynamic filtering was only proposed for matching processes with a THRESHOLD selection strategy within the final Select operator. In this thesis this is extended so that also a selection with a MAX-DELTA strategy can be the final selection operator. For that reason the P2 matching process contains a MAX-DELTA selection. As can be seen in Figure 8.4, again a significant improvement of execution time could be achieved. Surprisingly, the newly proposed dynamic filter for delta selection is able to also significantly reduce the execution time. In the XLARGE mapping cases an improvement with a factor 4 could be achieved which is significant. Note that the dynamic filtering by definition does not change the quality of the process. Hence any parallel matching system could apply dynamic filtering to improve performance. In this evaluation also the combination of dynamic filtering with threshold-based filtering further improved the performance. The maximum achieved improvement of factor 9 for extra-large mapping problems is promising.

When evaluating the matching process P1 and its rewritten counterparts the dynamic filtering that depends on the THRESHOLD strategy in the final Select operator did not improve the performance. This can be explained by a low selection threshold and the high number of matchers. P3 only consists of three matchers and the final threshold is higher. As expected the dynamic filtering now also improves the execution times. This improvement gets stronger with increasing schema sizes.

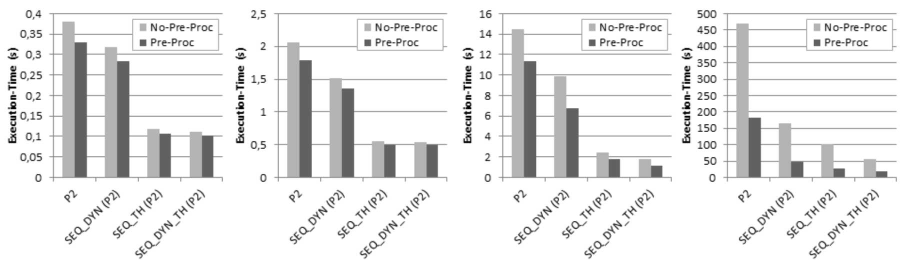


Figure 8.4: Performance evaluation of P2 and its rewritten processes

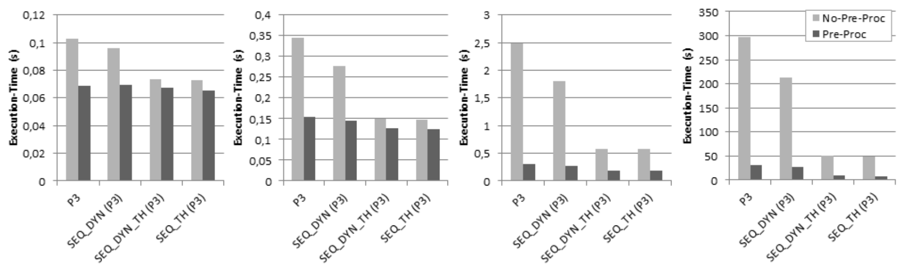


Figure 8.5: Performance evaluation of P3 and its rewritten processes

Still, the improvements are not as high as the dynamic filtering with MAX-DELTA selection. What can also be observed is that the improvements that can be achieved with precomputing token similarities do not get stronger with filtering. This can be explained by the fact that all three matchers, the Name, Namepath and Leaf matcher rely on the token preprocessing whereas some matchers in the P1 and P2 process do not benefit from that preprocessing like Instance, Sibling or Annotation matcher.

For all processes some execution time cannot be saved since the first matcher still has to compute the full cross product of source and target schema elements since Filter operators are applied afterwards. The token preprocessing already helps the first matcher to improve which explains why a combination of both techniques is beneficial.

## 8.2.2 Influence of Threshold on Execution Time

The threshold in the threshold-based filter rule in the last experiments was fixed to 0.2. In this subsection the relation between the threshold value and execution time is further investigated. For that purpose the P2 strategy is repeatedly rewritten and executed with the static filter-based rewrite rule but with increasing thresholds. The threshold of the filter conditions is changed from 0 (nothing is filtered) to 1 (everything is filtered). The effect is illustrated for two mapping problems in Figure 8.6.

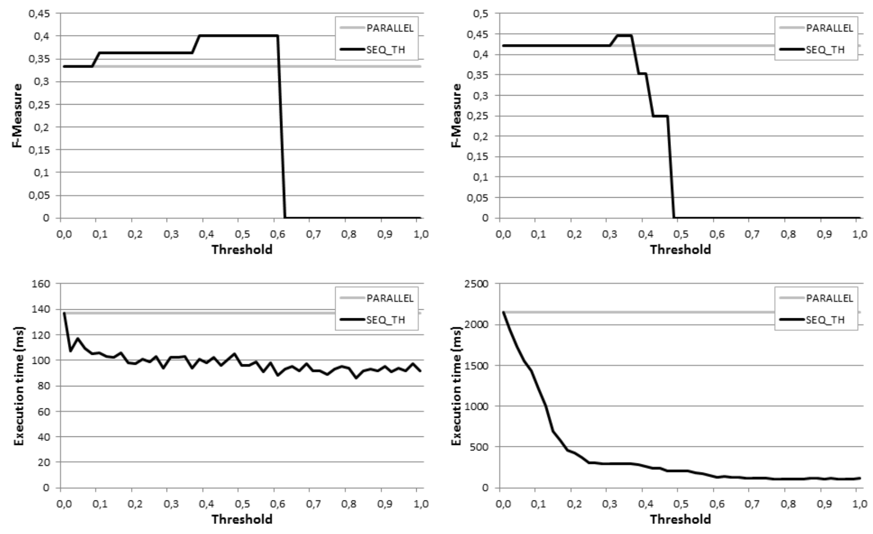


Figure 8.6: Execution time vs. quality

It explains the interplay of execution time and quality for a given mapping problem. For most cases the execution time decreases already significantly with low thresholds smaller 0.1. If thresholds increase too strongly $> 0.4$ relevant comparisons are pruned and the result quality deteriorates. This can be explained as follows: Many matchers produce many pairs with very low similarity e.g. $0.05$. Thus, a very high number of pairs can be pruned early. These pruned matches are very unlikely to contribute to the final result. With increasing threshold the number of pairs that can be pruned is significantly lower. Therefore performance improvements for higher thresholds are negligible. In some cases it can be observed that the filtering could also increase quality which is a positive side-effect. The Filter operator is able to prune out that noise, hence a small increase of the f-measure could be measured.

## 8.3 Evaluation of the Adaptive Rewrite-based Schema Matching Approach

After having evaluated the performance improvements that can be achieved by rewriting a matching process the rule-based matching process construction approach is now analyzed. At first, the robustness of the adaptive matching system is investigated. A matching system is robust, if it performs well for many different mapping problems and domains. Such a system not necessarily needs to perform better than a system that was tuned for a specific set of mappings. In order to test robustness, a comparison is made to a number of existing matching approaches that were described in Section 3.2. Each approach and its configuration is described briefly below:

- DEFAULT: A static matching process similar to the default strategy from COMA [35] is used. The DEFAULT process is computed by generating all possible combinations of all matchers of the matcher library. From these combinations a parallel matching process is created with an AVERAGE combination strategy and a Select operator. Different selection strategies, thresholds and MAX-DELTA parameters are tested. This creates a huge space of settings that is evaluated with the PO reference mappings. The strategy with the highest achieved f-measure is taken as best configuration. Computing the best configuration takes days to finish and highly depends on the number of available matchers and parameters of the selection strategy. The finally computed DEFAULT process consists of matchers NameWeighted, Path, Children, Leaf, Sibling and Type. The best selection strategy found was parameterized with MAX-DELTA and a delta value of 0.021 as well as a threshold of 0.5.

- META-LEVEL: A decision tree is learned as proposed with Meta Level Learning [49]. The approach includes schema features in the learning process to increase adaptivity of the finally computed decision tree. Again, the PO mappings serve as training set. The weka [173] library is used to train a J48 decision tree with a minimal number of objects per trained node of 10 to reduce overfitting. Also a re-sampling of the input mappings to increase importance of correct matches in the training set was considered. The learned tree from weka is taken and translated to a matching process without loss of information.

- OAEI-TUNED: As the third approach a matching process is manually constructed for the OAEI Benchmark. It is highly tuned to the OAEI specifics to achieve maximum possible f-measure. With appropriate selection thresholds, the precision can be tuned to be almost 1 which is a characteristic of most good performing matching systems that compete in the OAEI Benchmark.

- FALCON: The Falcon system [77] is taken as is. Since it relies on conditions with schema features it provides some adaptivity. An adapter was written that converts all schemas from the internal schema representation of the presented matching process execution framework into OWL. Mappings are converted into the RDF-based AlignmentAPI. Hierarchies can be converted to class or property hierarchies as proposed by [171] and both options returned similar results. By using the conversion to OWL it was possible to run Falcon on all schema mapping problems from the above data set.

- AGREEMENT: The AgreementMaker [30] system is also used for comparison. It computes schema features and learns which matching strategy to select from a set of good performing manually tuned strategies. Similar to Falcon the conversion to OWL and AlignmentAPI mappings was used to run the experiments. Since AgreementMaker relies on a learned model the original authors where consulted to provide the model that was used in the OAEI-evaluations from 2011.

- ADAPT: The rule-based approach from Chapter 7 together with the whole set of presented features and rules is compared to the other listed approaches and processes.

All these different approaches are executed to match the mappings of the test data set. As quality measure precision, recall and f-measure are used. In order to measure the effect of individual rules the adaptive system is executed with two additional settings. One setting always triggers rules as relevant if the input process matches to the structural pattern of a rule. In a second setting individual rules are excluded from the rule set. It can be expected that in many cases, excluding a rule will reduce quality. Also, there should be some differences measurable between always executing a rule and choosing a rule adaptively through a relevance condition.

The adaptive process generates different combinations of matchers, selection and combination strategies for each use case based on the analysis of the input schema features and intermediate results. In order to measure such heterogeneity, the distinct processes are counted. Since some rules adaptively compute parameters the probability that many similar processes are generated that only differ in a single parameter is high. For that reason it is tracked which rules trigger in each iteration. This creates a profile of triggered rules that could differ from case to case.

Finally, the value of the Monogamy feature for measuring the fitness of an iteration needs to be looked into. The number of iterations used for individual mapping cases is counted. The Monogamy-based termination condition is then replaced by a fixed iteration-counter assuming that always one, two or three iterations should be performed. Two iterations will certainly be the best iteration count since many matching systems first compute basic matches that are propagated in a second iteration. However, with Monogamy it should be possible to identify cases where more or less iterations would improve the quality.

### 8.3.1 Robustness of Overall Matching Approach

The results of comparing the different matching approaches are presented in Figure 8.7 (see Figure 8.8 and 8.9 for precision and recall). It shows the achieved f-measure for the individual data sets as well as a combined f-measure value (ALL) for all mapping problems which is the average of all average results for ES, OAEI, PO and IDOC. The ANATOMY case is not included in the average since the execution of the OAEI-TUNED and the META-LEVEL process was not possible due to memory problems in the current implementation of the matching process execution engine. The number of computed intermediate results in both processes simply consumed too much memory.

Overall, the ADAPT approach performs well in all mapping scenarios which was intended. It achieves best results in the ES and the IDOC data set and is close to the best f-measure values in all other scenarios. As expected, the adaptive approach is not able to be better individually than approaches that are tuned for specific mapping problems or domains. For instance, for matching the OAEI Benchmark

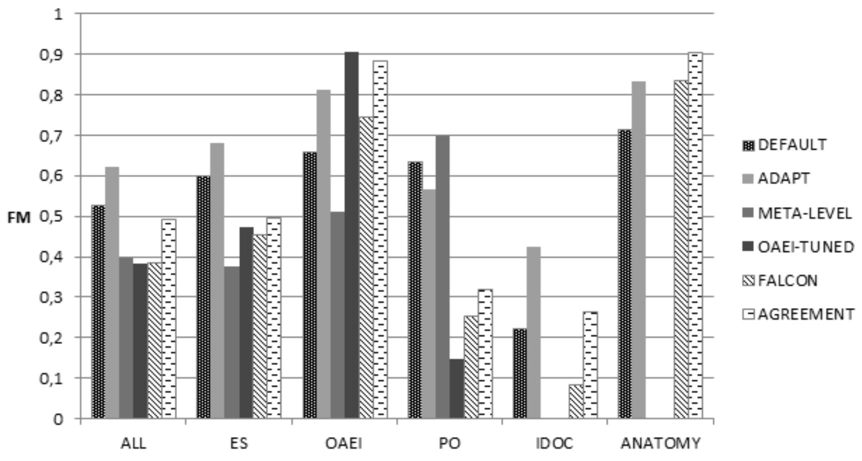


Figure 8.7: Comparing robustness (f-measure)

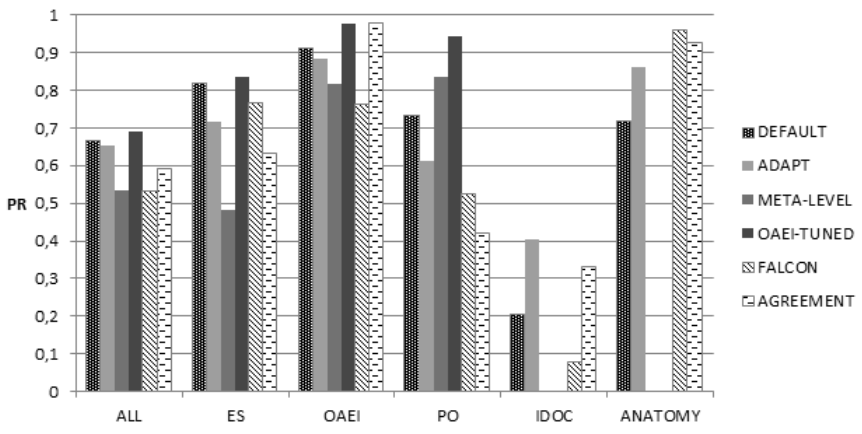


Figure 8.8: Comparing robustness (precision)

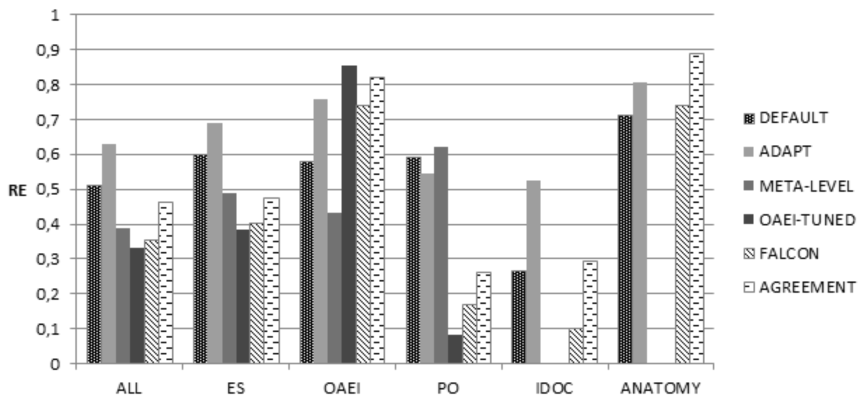


Figure 8.9: Comparing robustness (recall)

schemas, FALCON and the OAEI-TUNED approach perform better. However, systems that participate in the OAEI contest tend to rely on assumptions about the correct correspondences that do not hold in other data sets. For instance, if the basic matchers like Instance, Name or Annotation return a value close to one for a pair of elements, then the pair is in most cases a correct match. When Falcon participated in the first OAEI Contest 2005[58] it was among the best performing systems for the Benchmark track. In the years until 2010 the Benchmark track was yearly extended by additional synthetic mapping problems that are comparable to the ones from 2005. When Falcon participated the last time in 2010 it had problems to achieve top ranked results. Surprisingly, by correcting an error in the importer of Falcon within the evaluations in this thesis 4 percent higher FM values were achieved than Falcon could publish in the contest version from 2010 [53].

The DEFAULT process performs well with the PO data set since this set of mappings was used to compute a best configuration. In order to be comparable an Instance and Annotation matcher was added to the DEFAULT configuration. Surprisingly, this extended process also performed well in all other data sets which shows that the COMA approach of computing a default parallel matching process is quite strong. As could be shown in Section 2.5 the AVERAGE combination and the MAX-DELTA selection help to build a robust matching system.

The META-LEVEL process performed best on its training set but had severe problems in all other data sets. Here the problem of overfitting became obvious and also the original authors (Eckert et al.) already acknowledged that [49]. A number of parameters were tested to reduce the overfitting, but no bigger improvements could be achieved.

The OAEI-TUNED process returned best results with the Benchmark data set, as expected. However, it is not able to return good mapping results in the other use cases. As discussed already the OAEI-TUNED process strives for precision so that the precision is good in almost all data sets except for IDOC and ANATOMY where no mapping could be computed.

The FALCON system performed well in the ANATOMY test but was below average in all other cases. The matchers of Falcon strive for equality of name tokens which is a characteristic of the OAEI Benchmark and ANATOMY mappings for a number of correspondences. In particular within schema-based mapping cases FALCON has problems with the name matching. But also the structural matching approach of FALCON expects high structural similarity for being helpful. In the ANATOMY cases the structural matcher is automatically deselected through a condition on a structural similarity feature.

Finally, AgreementMaker was included in the evaluation since it was recently published as a novel adaptive matching technique. The AGREEMENT approach performed very good in the ANATOMY and OAEI cases where it was already used for. On schema-based mapping tasks AgreementMaker revealed some problems. But, it still managed to perform better than the Falcon system. What was interesting was the behavior of AgreementMaker in the ANATOMY case. AgreementMaker automatically selects the most appropriate matching process by profiling the input schemas. Since this automatic selection must be trained the original authors were contacted to provide the learned model that was used for the OAEI evaluations. However, by using their trained model a suboptimal matching process was chosen which resulted in an f-measure of 0.86. Assuming that a proper configuration of the AgreementMaker system could be achieved, the better f-measure above 0.9 was included in the charts.

### 8.3.2 Influence of Individual Rules

After having seen that the rule-based approach performs robust, the influence of individual rules needs to be assessed. For that purpose individual rules are removed from the rule library and adaptivity is switched off. In that setting rules always decide to be relevant. To reduce run-time and memory consumption the ANATOMY case was excluded from the further evaluation. The result of that experiment can be found in Figure 8.10. The “Adaptive” bar shows the average achieved f-measure of the system without any changes. “Always” reflects that an individual rule is chosen no matter what the relevance condition would recommend and “Never” removes the rule from the library. Additionally in Figure 8.11 the number of applications of individual rules is compared to the maximal number of possible rule applications. The second figure illustrates that the number of rule applications can be reduced to less than half of the possible applications as shown for rules like Weight-Name, Add-Statistics, Add-Leaf or Add-Annotation. The reasons are manifold. Not all schemas carry textual element annotations or instances that can be used for matching. For instance, the Add-Instance rule only triggers for half of the use-cases. This can be explained by the fact that not all schemas have instances to match on. A Leaf matcher and a Statistics matcher should only be used for mapping problems with high schema similarity which is also not given in all test cases. And finally, weighting tokens within the Name matcher can

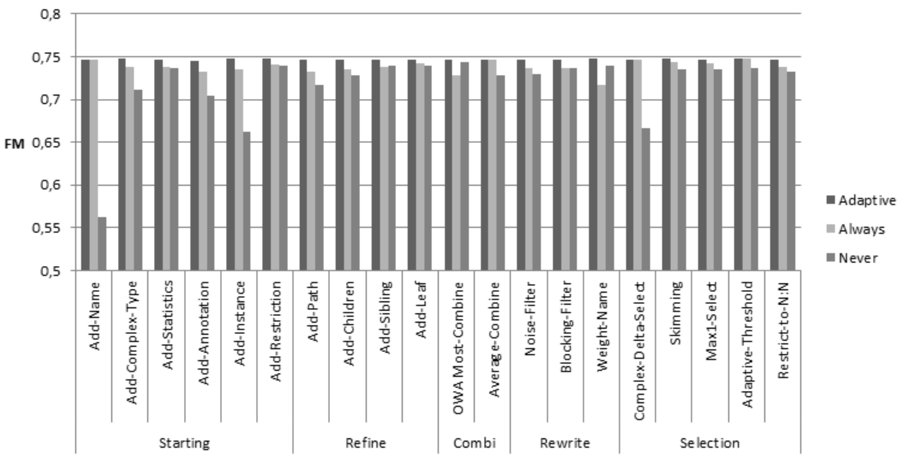


Figure 8.10: Evaluating individual rules

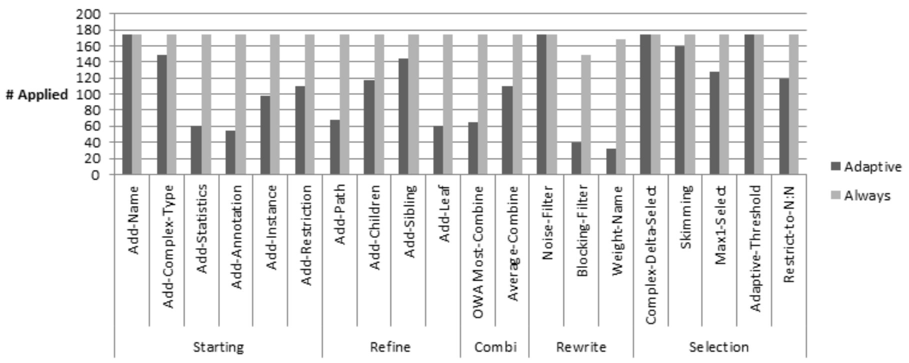


Figure 8.11: Number of applications of individual rules

only be done if element names consist of tokens and the Element-Token-Ratio feature value is low. Some rules are applied within almost all test cases. And some rules that add crucial operators like the Add-Name rule the number of rule applications is not reduced significantly.

Starting refine rules mainly add basic matchers. If rules like Add-Name, Add-Instance and Add-Annotation add matchers that are crucial for many mapping problems they should not be removed. However, always executing these rules may lead to smaller f-measure values which shows that adaptivity has some benefit. The benefit is smaller for rules that help to solve almost all mapping problems like Add-Name and Add-Complex-Type. In contrast to the Add-Name rule, the Add-Restriction and Add-Statistics rule do only significantly contribute to a small set of mapping problems. In particular in the OAEI cases a Restriction matcher is helpful and is able to identify additional correspondences. Still, the profit of choosing a rule adaptively in comparison to always executing it seems rather small. This can be explained by the robustness of the AVERAGE combination and MAX-DELTA selection which could already be shown in the pre-evaluation from Section 2.5. Still, if too many matchers provide 0-similarity input the combination and subsequent selections do run into problems. All small improvements from choosing rules adaptively do (partially) add up to improve the quality for the whole system. Moreover, adaptivity could help to reduce execution times of a system since unnecessary matcher executions can be saved.

The outcome differs a bit with refine rules that add structural matchers. Not executing a structural matcher or always executing it has a smaller effect on f-measure. Surprisingly, the Add-Path rule is only rarely chosen. This can be explained by the many Benchmark cases where a Path matcher is not helpful. If the rule triggers it contributes to improve the quality. Again, the combination is robust so that always triggering the Add-Path rule does not significantly reduce result quality. In contrast to the Add-Path rule the Add-Sibling rule triggers very often. Obviously, the distribution of siblings that can be computed from intermediate similarity matrices rarely happens. Still the relevance condition is able to filter out those cases where the Add-Sibling cannot contribute which can be seen from the comparison to the “Always” bar. The Add-Leaf rule is rarely selected. Its influence on quality is limited. Thus, the relevance condition that solely relies on structural similarity still needs improvement.

The OWA-Most combination rule is also rarely chosen. If chosen, it only little contributes to the overall result quality. From the measured numbers one could conclude to not include the OWA-Most rule at all. As discussed the Average rule is always chosen as fallback.

Another contribution to quality comes with rewrite rules. The Noise-Filter rule is often chosen, since most matchers produce noise. The individual effects are small but they add to a measureable average improvement. The Blocking-Filter rule is rarely chosen due to a very pessimistic relevance function. If wrongly blocked correspondences can be found in the high precision selection result, the rule is not applied. Moreover, the rule can only be chosen as often as there is a Type matcher present in the current matching process. Due to the rare applications of the Blocking-Filter rule, the average quality improvement is small. Similarly, the Weight-Name rule is chosen rarely. The OAEI test cases do not contain multi-token element names and they make up more than half of all test cases. The relevance condition of the Weight-Name rule seems appropriate to select the most relevant cases where improvements can be achieved. This can be seen by comparing to the “Never” bar. Always executing the rule seems unpractical since token weighting could also decrease quality if applied for the wrong cases.

Finally, the selection rules also help to improve the quality of the adaptive matching system. The Select-Complex-Delta rule is always chosen, since it is used as the default fallback selection approach. It adds a complex combination of Select, Filter and Combine operators to the process which shows to be quite robust. The Skimming rule only contributes with small improvements of mainly recall. The Max1-Select rule is chosen in cases where 1:1 mappings are assumed and the relevance condition computed to

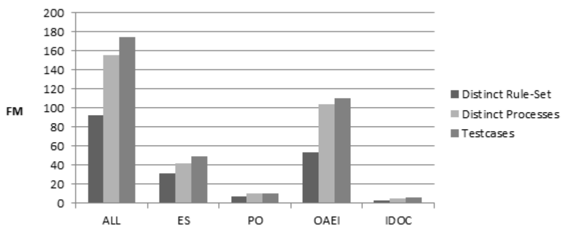


Figure 8.12: Variance of generated processes

true. The Adaptive-Threshold rule is always triggered and achieves small average effects. In some individual cases, up to 4 percent improvement could be achieved. And finally, the Restrict-to-N:N rule often triggers. Its relevance condition relies on the average number of repeating components N and M in both input schemas. It then adds a matcher that restricts multi-matches to N:M matches which can improve precision. In many cases an EXACT selection is added that restricts to 1:1 matches. In particular in the OAEI cases this can help to improve the precision.

### 8.3.3 Process Heterogeneity

The number of rule applications already shows that for each mapping problem different rules trigger. In order to show that only a small set of similar matching processes is generated the number of distinct processes is computed. As can be seen in Figure 8.12 almost every test case of the 175 cases gets a unique process generated by the adaptive execution. However, due to operators that compute thresholds or other parameters only minor differences might exist between processes. If only rule applications are tracked per test case still 90 different processes are generated. Due to the heterogeneity of test cases in the ES set more variance in generated processes can be observed. The OAEI-set contains many similar problems with only minor changes in specific schema attributes. Therefore, less different processes are generated for those use-cases.

### 8.3.4 Monogamy-based Termination of Iterations

Finally, the iteration condition that is based on the Monogamy feature is analyzed. For that purpose the adaptive system is executed with a fixed number of iterations between one and four iterations for the ES, PO and OAEI dataset. The result is compared to the adaptive approach based on Monogamy. Moreover, the upper limit (MAX) is computed by taking the maximal achievable f-measure values from each test case to identify the optimal solution. The result of Figure 8.13 illustrates that the best iteration count is two in most cases as expected. However, there are cases where more iterations improve the result. The major outcome is, that adaptively terminating iterations is better than the fixed iteration count. Moreover, the adaptive solution is close to the optimal solution which is encouraging.

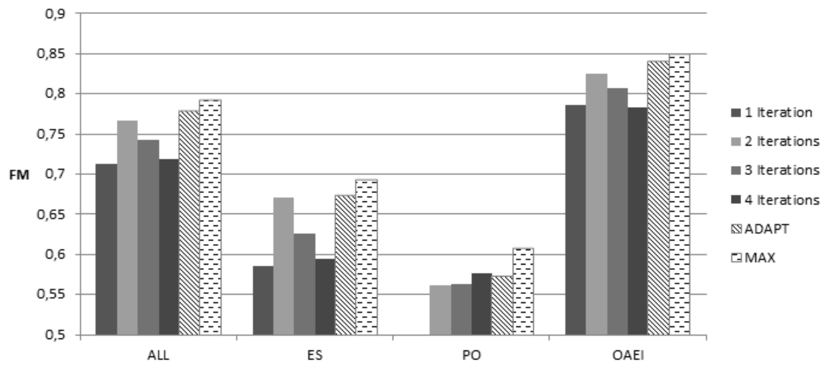


Figure 8.13: Evaluating the termination condition to stop iterations

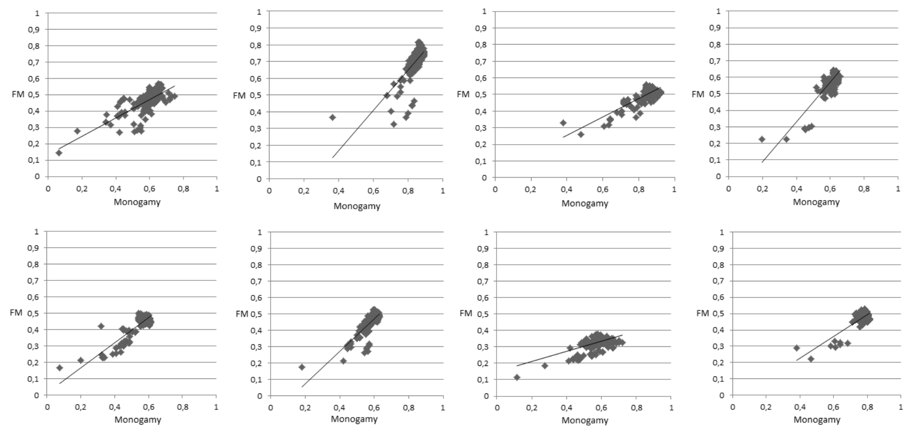


Figure 8.14: Correlation of Monogamy with f-measure on the PO set

In order to further back the application of Monogamy additional experiments were done with finding best matcher combinations which is similar to evaluating the result of a refine phase. Eight medium sized mapping cases from the PO set were taken and all possible matcher combinations were computed. Parallel matching processes were constructed by using the MAX-DELTA selection and the AVERAGE combination. Each process is executed on the individual cases and the f-measure and Monogamy value is measured. Both values are plotted in a chart for each mapping problem and process which are shown in Figure 8.14. Obviously, there is a strong correlation of the f-measure with the computed Monogamy value. In cases where many multi-matches are contained in the gold standard, the correlation is less strong. This can be explained by the fact that Monogamy favors 1:1 mappings over multi-mappings.

## 8.4 Summary of Evaluation

The evaluation proves that both presented approaches are effective with a number of heterogeneous mapping problems. It could be shown that the rewriting of matching processes with Filter operators significantly reduces execution times in particular when the final Select operator relies on a MAX-DELTA selection strategy. Dynamic filter conditions are particularly interesting since they are able to reduce execution times without changing the quality of the process which is similar to rewrite-based performance optimization in database queries (predicate push-down). However, they were not always effective, in particular if a large number of matchers is used.

The cost model seems sufficient for the filter-based rules and their performance improvement goal. The rewrite-based approach on the process level is orthogonal to existing preprocessing techniques like precomputing token similarities. Also clustering strategies could be used in addition to further improve the execution times significantly. Filter-based rewrite rules have initially not been intended to improve the quality of the process. But, rewrite rules allow a user to execute more matchers within a given time-frame which then could improve quality.

For improving matching quality, the adaptive rewrite-based process construction approach has proven to be effective. The system robustly constructs a good performing matching process that fits to the specifics of the input mapping problem. The influence of individual rules differs. Some rules could always trigger without influencing quality, whereas others improve the quality by beeing triggered adaptively. Surprisingly, choosing appropriate combination rules that select optimal combination strategies is still problematic. Finding a generic relevance condition and appropriate matrix similarity features still remains open for future work. Finally, the evaluation of the termination condition shows that Monogamy is appropriate for that task. But, its application is not limited to that use-case since it well correlates with f-measure for 1:1 mapping cases. It could also be used for finding good matcher combinations or for finding optimal thresholds of a THRESHOLD selection strategy.

While evaluating large mapping problems, major issues arose with memory usage. This is partly an implementation issue of the matching processes that store intermediate results as similarity matrices. However, some matchers also store some information internally that requires more memory with increasing schema sizes. This could certainly in future be improved by using less memory-intensive data structures. Also the run-times of the adaptive system are higher than the system that executes a performance-optimized matching process. This could be tackled by performing the adaptive process construction on smaller pairs of sub-schemas. The constructed process can then be rewritten and executed on the complete schema mapping problem.

# Part V

# Summary and Outlook

# Chapter 9

# Summary and Outlook

## 9.1 Summary

The thesis investigated the problem of configuring schema matching systems that are used to partially automate the process of finding mappings between schemas. In the first part fundamental concepts of schema matching where introduced and an overview to the existing body of work was given. In particular approaches that support manual tuning and that (partially) automate the configuration and construction of schema matching systems were reviewed. The second part of the thesis then introduced a new matching process model that supports adaptivity as well as tools and a framework for manual matching process construction. The third part finally introduced novel rule-based techniques for automating the construction and configuration of matching processes.

### 9.1.1 Process-based Schema Matching

Initially, the concept of adaptive matching processes was defined. The presented matching process model formed the basis for subsequent contributions within this thesis. Matching processes can be used to model the execution order of operators in a schema matching system. Operators were presented for im- and exporting schemas and mappings, for matching, combination and selection as well as for filtering match comparisons which can be used for increasing the run-time performance of matching processes. Moreover, some control structures like Condition, For-Each and Loop were presented. In particular, the Condition operator together with a number of newly introduced schema and matrix features can be used to implement adaptive behavior of a matching system and to increase its robustness. A number of features were described that can capture characteristics of mappings and input schemas. A core feature that was presented was the Monogamy feature that sometimes correlates well with the final mapping quality for mapping results without having a reference mapping at hand. This works primarily for one-to-one mapping problems. With the help of the presented operators, the internal workflows of most existing matching systems can be represented. An execution framework was presented that is able to execute matching processes. Through a plug-in architecture, components from different matching systems can be integrated and matching processes consisting of parts from multiple systems can be constructed which fosters reuse of existing matching components.

In order to ease development and configuration of matching processes a graphical modeling tool was presented. It allows a user to construct matching processes by using a drag and drop metaphor. Moreover, it visually supports the tuning process by providing novel visualizations for analyzing intermediate mapping results. The tool also offers means to evaluate and compare multiple matching processes with provided reference mappings to support the user in selecting appropriate matchers, selection or combination strategies. From analyzing internal matching processes of existing systems so-called matching process design patterns where identified and their advantages and disadvantages where compared.

### 9.1.2 Automatic Configuration and Construction of Matching Processes

Even with tool support, the configuration of matching processes remains a complex task. In the second part of the thesis a novel approach was presented that relies on so-called rewrite rules, features and filter strategies. Initially such rewrite rules were only used for improving the performance of a schema matching process. By sequentializing parallel matching processes with filter-based rewrite rules significant run-time performance improvements (up to a factor of 9) could be achieved. In particular together with a dynamic filter strategy improvements were achieved without changing the quality of a schema matching process.

A second contribution adopted the concept of rewrite rules for constructing an adaptive schema matching system. It applies rewrite rules to automatically construct matching processes tailored to given mapping problems. These rewrite rules rely on analyzing the input schemas and intermediate results while executing a process and rewrite the process to better fit to the problem at hand. For that purpose the previously introduced features were used. Based on these features, various rewrite rules for adding matchers, combination or selection operators to a process or for rewriting a given process were presented. The evaluation showed that the proposed approach can be used to construct a robust schema matching system that is able to achieve good matching quality across a number of heterogeneous use-cases. The final system is self-configuring so that is does not need other input than the input schemas to compute a mapping. However, additional synonym dictionaries or thesauri could be given as parameters to existing matchers.

## 9.2 Outlook

Within the thesis a number of further research directions could be identified that either extend the proposed solutions or provide potential for increasing the quality of semi-automatic schema matching systems. This section describes four possible directions in more detail.

**Distribution and Parallelization** The evaluation of run-time performance optimizations were promising. But still for large size mapping problems in particular in life-sciences the execution times need to be further reduced significantly. In particular, memory consumption is a major issue in schema matching when many large sized similarity matrices are produced as intermediate results. This problem can partly be alleviated by alternative data structures for intermediate results that only store correspondences above a given threshold [68] or by partitioning the input schemas into smaller blocks [77]. But still, approaches to increase performance by distributing the execution of matchers onto multiple nodes would be necessary. The authors of [69] speak of inter- and intra-matcher parallelization. Intra-matcher parallelization distributes comparisons of a single matcher onto multiple nodes whereas inter-matcher parallelization distributes complete matchers on individual nodes. The matching process model could be extended by performance aspects such as marking operators for inter- or intra-operator parallelization which describes how to distribute certain operators in the process. The matching process could then serve as an input script for parallelization.

**Further Rules for Adaptive Combination** The adaptive rewrite-approach relies on a number of different rules for choosing appropriate matchers. Yet, only two combination rules are part of the rule library with the Average-Combination rule as fallback that always triggers if the other option (OWA-Most-Combination rule) is not relevant. The evaluations for that rule were not yet convincing. Given the high number of existing combination approaches [137] some work still needs be done to identify appropriate matrix similarity features that can be used for relevance conditions of combination rules. Also other types of rules can still be added. The set of rewrite rules is still limited and could be extended with further matching knowledge by other researchers. For instance, adaptive matching process construction mostly relies on schema-level information and matrix features. Since the metadata provided in schemas is often weak instances should be increasingly used. However, for matching instances a number of different approaches where presented in literature, choosing the most appropriate one for given instances is challenging and could be done in analogy to the presented feature-based approach.

**Standardization and Implementation Reuse** When analyzing related work on schema matching it became obvious that more effort needs to be invested into a standardization of existing matching system components. The proposed matching process model with its set of operators could serve as a starting point for such standardization. Current initiatives try to compare whole matching systems with each other. For instance, the SEALs platform offers an interface to a Matching system that can be implemented by participants. In an evaluation phase all participating systems are called through that interface and a diverse set of mapping problems are executed. However, these evaluations do not give evidence about what actually makes up a well performing system since that depends on a number of factors like selection strategies, combination strategies conditions and matchers. In order to foster reuse the evaluations should be done by operator so that only selections, combinations or specific types of matchers are compared separately.

**Change to Top-N Results** Most schema matching systems strive for a single shot matching approach where one complete mapping recommendation is computed once. Within all evaluations of this thesis there were always some test cases with f-measure values lower than 0.5 so that the effort to correct computed mappings is higher than manually defining a mapping from scratch. But, this approach does not fit to the need of a mapping designer that manually defines mappings. What is needed is an incremental matching approach that computes multiple matching candidates for selected elements. The set of candidates can be called Top-N sets. Within preliminary experiments it was possible to show that 90% of the correct matches where included in the Top-10 result for many mapping problems. Hence, even for hard to match problems schema matching could help if matching systems and mapping UIs would support Top-N results. Top-N matching creates a number of new interesting problems. When computing new Top-N sets, candidates that were already selected or rejected for other source elements can be incorporated for computing further candidates. In preparatory work for this thesis further opportunities were found to optimize the Top-N selection strategy [138]. The idea is to reduce the overlap of Top-N sets which then could increase recall. Overlap refers to candidates that are contained in Top-N sets of many source elements, even though an element can only be a match for single elements. By reducing the degree of overlap additional candidates can be added to Top-N-sets. The Top-N problem also involves changes to user interfaces on how to best present a set of candidates and how to simplify selection of the correct candidate. Moreover, the adaptive rewrite-based matching system could be changed to optimize towards a Top-N result. In that case, combination, selection and relevance conditions could favor reduced overlap or other to be defined features of Top-N mappings.

# Appendix A

# Feature Collection

## A.1 Schema Features

### A.1.1 String-Meaningfulness Feature

Some string similarity measures rely on language models or directly consume background thesauri for computing string similarities such as the Wordnet matcher from RiMOM. In the OAEI Benchmarks, some schemas were artificially changed by scrambling labels. Based on a Wordnet lookup, RiMOM was able to entirely skip any name matching in such cases. However, the Wordnet lookup is very restrictive with regards to English. Also terms that are technical but carry a meaning would not be found in Wordnet. In order to be able to measure how meaningful names or annotations of a schema are, a generic approach is presented. It is based on querying a search engine such as Google[1]. The assumption is that Google-queries have a high total result number if a term is meaningful. Meaningful does not necessarily mean that a term is natural language. Many search results could also imply that many people use a term and assign a meaning to it. Certainly, this assumption does not hold for all terms used in a schema. But, on average for all terms of a schema a meaningfulness value can be derived. The String-Meaningfulness feature is defined as follows.

$$StringMeaningfulness\,(S) = \frac{median\,(sg(x_1), \ldots, sg(x_n))}{max} \tag{A.1}$$

with $sg(x)$ being the total number of search results when searching with the search key $x$. The term $x$ can be a name or an annotation of an element $e$. The totals of different search terms vary strongly. For that reason the median is computed from all search totals of all elements of a schema. For big-sized schemas, searching for every term could be time-consuming. Therefore, a sample of terms (100) is taken and a median is computed. The resulting feature values vary slightly with each new computation which is acceptable in most use-cases.

[1]www.google.com

### A.1.2 Element-Token-Ratio Feature

Element-Token-Ratio is a feature which is very specific for term/token-based schema element names. In real-world use-cases it can be observed that schema elements are often named or annotated with a combination of a very small set of terms. Schema designers sometimes only use a small set of terms and concatenate them to name schema elements. A string similarity measure like tri-gram would have problems to cope with the ambiguity introduced from such reuse of terms and tokens for different schema element names or annotations. For identifying such schemas the Element-Token-Ratio feature can help. It relies on a function $tokenize(q)$ that collects all terms or tokens of a given string $q$. Then, all tokens $Q$ from attribute $x$ of all schema elements $\{s_{1,}, \ldots, s_n\}$ of a schema $S$ are collected:

$$Q = tokenize(s_1.x) \cup \ldots \cup tokenize(s_n.x) \tag{A.2}$$

For each token the ratio of occurrences in each schema element is summarized. The sum of these ratios is then divided by the number of all tokens of a schema. The Element-Token-Ratio for an attribute $x$ can then be written as follows:

$$ElementTokenRatio_x(S) = \frac{\sum_{t \in Q} \frac{1}{|\{a \in S \mid t \in tokenize(a.x)\}|}}{|Q|} \tag{A.3}$$

If all tokens of a schema are only used once within a schema then the Element-Token-Ratio value is 1. The more reuse of tokens is done, the smaller the Element-Token-Ratio value gets. With the help of the Element-Token-Ratio feature, an appropriate matcher can be chosen that tries to include the relative importance of terms into the computation of name or annotation similarities.

### A.1.3 Repeating-Elements Feature

The Element-Token-Ratio feature would also have high value if whole elements or fragments of a schema are reused which is not related to the problem of reusing tokens. For that purpose, the Repeating-Elements feature was introduced. It measures how often element names and their content are repeated within a schema. In particular in XSD schemas types are often reused which creates ambiguity and high values in the Repeating-Elements feature.

$$RepeatingElements\,(S) = \frac{|distinct(\{s_1, \ldots, s_n\})|}{|S|} \tag{A.4}$$

### A.1.4 Repeating-Fragments Feature

The Repeating-Fragments feature measures how often small fragments and their properties are repeated within a schema. For computing the feature value the smallest fragment possible, a pair of nodes in parent/child relation $PCPair$, is considered. If such pairs occur repeatedly within a schema, reuse of fragments can be assumed.

$$RepeatingFragments(S) = \frac{|distinct\,(PCPairs(S))\,|}{|PCPairs(S)|} \tag{A.5}$$

### A.1.5 Schema-Depth Feature

Many schemas are tree-structured or are at least tree-like, so that by removing few edges a minimal spanning tree could be computed. From a tree, structural features can be computed such as the average path length. If paths are short the schema is not deeply structured and structural matchers will have lesser influence.

$$leaves(S) = \{s \in S \mid s.children = \emptyset\} \tag{A.6}$$

$$SchemaDepth(S) = \frac{\frac{\sum_{l \in leaves(S)} |path(l)|}{|leaves(S)|}}{maxDepth} \tag{A.7}$$

The Schema-Depth feature has a similar goal like Inheritance Richness from [162] which measures the average number of subclasses per class.

### A.1.6 Path-Variance Feature

In particular when matching relational or XSD schemas the Path matcher is often very effective. It is a structural matcher that considers the path similarity to compute the similarity of schema elements. However, not for every schema the Path matcher is appropriate to be used. Therefore, a Path-Variance feature is proposed. It identifies all leaf elements of a schema and enumerates their paths up the parents (parentPath) of the individual leaf elements. The ratio of the number of all distinct paths and the number of all leafs leads the Path-Variance feature value:

$$PathVariance\,(S) = \frac{\left|\bigcup_{l \in leaves(S)} parentPath(l)\right|}{|leaves(S)|} \tag{A.8}$$

If all paths are distinct then a high path variance is given. In such cases a Path matcher can well disambiguate. If only a single path leads to all leave elements as in flat relational schemas, path matching will not perform well.

## A.2 Schema Similarity Features

### A.2.1 Feature-Similarity and Average

Two things have to be taken into account when combining feature values. First, the difference of the feature values for the source and the target schema can be computed and second the average is a good indicator. Getting the average value is trivial. For computing the difference Cruz et al. [30] proposed to compute a so-called Feature-Similarity (FS) value:

$$FS(f_1, f_2) = \frac{min(f_1, f_2)}{max(f_1, f_2)[log(max(f_1, f_2) - min(f_1, f_2) + 1) + 1]} \tag{A.9}$$

The more similar the two input feature values are the closer the FS-value is to 1. Depending on the use case either the Average value or the Feature-Similarity value can be used.

### A.2.2 Similar-Language Feature

(see Section )

The Similar-Language feature computes the probability of a text $\{t_1, \dots, t_n\}$ being of some given language L with the help of a language identification function $li_L : \{t_1, \dots, t_n\} \rightarrow [0, 1]$. All strings of an attribute $x$ from all elements of the schema $S$ are collected and the probability of being of language L is computed.

$$language_L(S) = li_L(\{s_1.x, \dots, s_n.x\}) \tag{A.10}$$

This function can be extended to return the top-N languages a text could belong to together with their probabilities. By comparing the top 3 proposed languages for the source schema with the languages proposed for the target schema the *Similar-Language* feature value can be computed.

### A.2.3 Structural-Similarity Feature

Structural-Similarity is a measure to compute how similar the structural shapes of two schemas are. A high structural similarity is an indicator to increase the relevance of structure-based matchers. In RiMOM, a feature that was called Structural Similarity Factor was used to compute structural similarity of two schemas. Two schema elements are similar if their child concept counts are similar and their paths to root have similar length. The number of so-called common concepts is divided by the maximum number of non-leaf elements in the source or target schema. Later, the RiMOM feature was extended within UFOMe. UFOMe refers to the element statistics as Intrinsic Information Content $lc(s)$ of a schema element $s$.

$$lc(s) = 1 - \frac{log(Sub(s) + 1)}{log(|S|)} \tag{A.11}$$

where $Sub(s)$ computes the number of child nodes of an element s.

The number of common concepts and the Structural Similarity feature can then be computed as follows:

$$common = \{(s, t) \in S \times T \,|\, \exists(s, i) : |path(s)| = |path(i)| \wedge lc(s) - lc(t) < th\} \tag{A.12}$$

$$StructuralSimilarity = \frac{|common|}{min(|S|, |T|)} \tag{A.13}$$

Within this thesis, this feature was reused since it showed to be a good indicator for selecting matchers.

## A.3 Matrix Features

### A.3.1 Selectivity Feature

Selectivity tries to evaluate the confidence of a result matrix that was computed by a matcher or subprocess. It computes the distance of the top-1 entry in a row or column to the next highest entry in the same row and column. The rationale is that a high distance of the best candidate match to the next possible matches implies that the candidate match is certain. A low distance on the other hand shows more uncertainty. The Selectivity feature was formalized by Eberius [48] as follows. For a vector V sorted in descending order (so that $V_0$ is the similarity of the best, and $V_1$ of the next best candidate) we compute the selectivity of the vector as:

$$selectivity(V) = \begin{cases} 0 & if\ V_0 = 0, \\ V_0 - V_1 & else. \end{cases} \tag{A.14}$$

For a similarity matrix $SM$ with $N * M$ entries the selectivity value can be computed as follows:

$$Selectivity(SM) = \frac{\sum_{i=1}^{N} selectivity(SM_{i,*}) + \sum_{j=1}^{M} selectivity(SM_{*,j})}{|N| + |M|} \tag{A.15}$$

$SM_{i,*}$ refers to the vector of values in the i-th row of a similarity matrix SM. All selectivities of rows and columns are summed up and divided by the number of candidate entries in the matrix. If the selectivity is very low, the likelihood that after a selection many 1:N, N:1 or N:M matches will result is very high. For example, a high selectivity could indicate to use a MAX-1 selection when a selection strategy needs to be defined.

### A.3.2 Cross-Matches Feature

The Cross-Matches feature computes how structurally consistent a computed mapping is, i.e. how structurally close the matching target elements and source elements are. A low structural consistency is an indicator for low precision mappings. For computing the Cross-Matches feature the ratio of the number of matches that are crossed by some other match to the total number of matches above a given threshold $th$ is computed.

$$CrossMatches(SM,S,T) = \frac{|\{(i,j) \in SM | sim(i,j) > th \wedge \exists (x,y) \in SM \, with \, x \in leaves(j,S), y \in path(i,T)\}|}{|\{(i,j) \in SM | sim(i,j) > th\}|} \tag{A.16}$$

The function $leaves(j,S)$ computes the set of leaves of an element $j$. The function $path(i,T)$ computes the set of elements in the path of an element $i$.

### A.3.3 Node-Position Feature

The Node-Position feature also computes how structurally consistent a computed similarity matrix is. Assuming overall structural similarity the path-length of matching element pairs should be similar or differ in the range of a given delta. The Node-Position feature is then defined as follows:

$$NodePosition(SM,S,T) = \frac{|\{(i,j) \in SM | sim(i,j) > th \wedge |path(i,S)| - |path(j,T)| > delta\}|}{|\{(i,j) \in SM | sim(i,j) > th\}|} \tag{A.17}$$

If the path-length differs significantly for many matches, then either the structural similarity is low or the precision of the matrix is low.

### A.3.4 Multi-Matches Feature

The Multi-Matches feature represents the ratio of N:M matches to the number of 1:1 matches within a similarity matrix. This feature can be used to compute the relevance of rules that reduce the multi-matches in order to increase quality. First, all 1:1 matches $O(SM)$ are computed for a similarity matrix SM.

$$\begin{aligned} O(SM) = \{(s,t) \in SM \mid sim\,(s,t) > 0 \wedge (\nexists\,(s,j) \in SM : sim\,(s,j) > 0 \wedge j \neq t) \\ \wedge (\nexists\,(i,t) \in SM : sim\,(i,t) > 0 \wedge i \neq t)\} \end{aligned} \tag{A.18}$$

Then the Multi-Matches feature is computed as follows:

$$MultiMatches(SM) = \frac{|O(SM)|}{|\{(i,j) \in SM | sim(i,j) > 0 \,\wedge\, (i,j) \notin O(SM)\}|} \tag{A.19}$$

### A.3.5 Sibling-Distribution Feature

A similarity matrix and the correspondences computed from it can give a good indication about structural similarities. For instance, the Sibling matcher relies on the siblings of a source and target element to derive a similarity value. However, if the structure is dissimilar then the Sibling matcher would compute misleading results. With respect to the siblings of elements a so-called distribution of matches can be observed. If two correspondences link sibling elements in the source schema and also link to sibling elements in the target schema then a structural similarity of the respective fragment can be assumed. If many such sibling correspondences exist within a mapping then a high structural similarity can also be assumed for the mapping. Based on that observation a Sibling-Distribution feature can be defined as follows. All mapping pairs $siblingPairs$ above a given threshold are enumerated that possess at least one sibling correspondence.

$$\begin{aligned} siblingPairs = \{(i,j) \in SM | sim(i,j) > th \wedge \exists (x,y) \in SM, \\ with\, sim(x,y) > th,\ x \in siblings(i,S), y \in siblings(j,T)\} \end{aligned} \tag{A.20}$$

Sibling-Distribution can then be computed from the following ratio:

$$SiblingDistribution(SM,S,T) = 1 - \frac{|siblingPairs|}{|\{(i,j) \in SM | sim(i,j) > th\}|} \tag{A.21}$$

High values of Sibling Distribution imply a low structural similarity. If Sibling Distribution is low then most correspondences are sibling correspondences. In such cases, applying a Sibling matcher could be useful.

## A.4 Matrix Similarity Features

### A.4.1 Commonality Feature

Input similarity matrices $SM_1, ..., SM_x$ can be analyzed according to their commonalities. The Commonality feature rates how common a set of matrices is. In order to measure the commonalities the most trustful mapping pairs are extracted from each input matrix $SM_y$. Trustful pairs with high precision can be acquired by applying a selection with backward and forward direction like $select_{delta,both}(SM_y)$ and an additional EXACT restriction to the 1:1 matches of the selected matrix. Then all selected matrices $Q_y$ are combined with a MAX combination to Q.

Then, a vote $v$ for each element pair $(i,j)$ can be computed that expresses how strong the input matrices agree on a pair. The sum of votes per pair is divided by the maximal possible sum of votes.

$$Q_y = select_{exact}\left(select_{delta,both}(SM_y)\right) \tag{A.22}$$

$$Q = combine_{MAX}(Q_1, ...Q_X) \tag{A.23}$$

$$v(i,j) = \frac{\sum_{y=1}^{X} |\{(i,j) \in Q_y \mid sim(i,j) > 0)\}|}{X} \tag{A.24}$$

$$Commonality(SM_1, \ldots, SM_{|Q|}) = \frac{\sum_{(i,j) \in Q \mid sim(i,j) > 0} v(i,j)}{|\{(i,j) \in Q \mid sim(i,j) > 0\}|} \tag{A.25}$$

High values of the Commonality feature are computed if all matrices agree on a pair whereas no agreement on any pair would lead to small values.

### A.4.2 Complementarity Feature

The Complementarity feature also relies on filtering trustful pairs from every input matrix. The ratio of conflicting pairs to all pairs $All$ computes the Complementarity value.

$$\begin{aligned} conflicting(SM) = \quad & \qquad (A.26) \\ \{(i,j) \in All \mid (\exists(i,y) : (i,y) \in All \wedge sim(i,y) > 0 \wedge y \neq j) \\ \vee(\exists(x,j) : (x,j) \in All \wedge sim(x,j) > 0 \wedge x \neq i)\} \end{aligned}$$

$$Complementarity(SM) = \frac{|conflicting(SM)|}{|All|} \tag{A.27}$$

If input matrices are not common and also do not complement each other, then not all matrices should be considered for the subsequent combination or some should get a lower weight assigned.

# Appendix B

# Rule Collection

## B.1 Starting Refine Rules

### B.1.1 Add-Statistics Rule

The Add-Statistics-Matcher rule adds a combination of a Parent-Count and a Child-Count matcher to the matching process. The combined Match operators are shown within the action pattern below the bar of Figure B.1. For computing the relevance of the Add-Statistics rule, the Structural-Similarity feature can be used. This feature solely derives its value from the element path and child count statistics of source and target elements. Thus, it serves well as an indicator of when to use the statistics matcher. The resulting relevance condition is therefor rather simple to formulate:

$$Condition_{Statistics} = StructuralSimilarity(S, T) > 0.5 \tag{B.1}$$

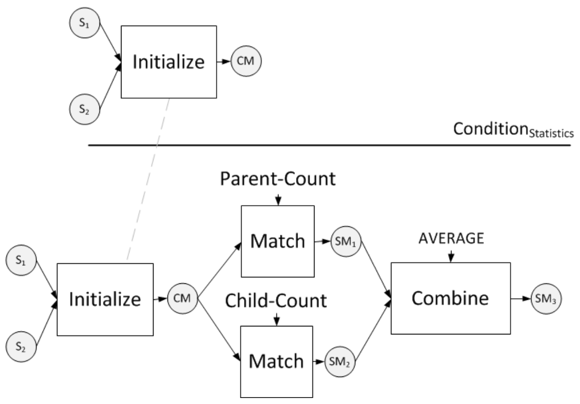


Figure B.1: Add-Statistics rule

### B.1.2 Add-Annotation Rule

The Add-Annotation rule shown in Figure B.2 adds an Annotation matcher to the matching process.

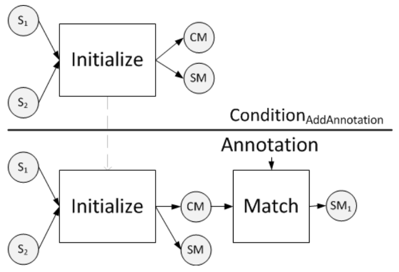


Figure B.2: Add-Annotation rule

The Annotation matcher relies on the Soft-TFIDF [29] measure to compute the similarity of textual annotations of schema elements. Similar to other starting rules the Add-Annotation rule relies on the existence and variance of annotations within the source and target schema. Moreover, some overlap of annotation tokens should be measurable in order to be able to expect trustful similarity values from the Annotation matcher. However, the overlap can be smaller than the overlap needed for the Name matcher. This can be explained by the length of possible annotations that could contain multiple lines of text per element. This results in the following relevance condition:

$$\begin{aligned} Condition_{AddAnnotation} = \\ \neg(AnnotationExistence(S) \leq 0 \lor AnnotationExistence(T) \leq 0) \\ \land(AnnotationVariance(S) > 0.2 \land AnnotationVariance(T) > 0.2) \\ \land AVG(AnnotationTokenOverlap, S, T) > 0.05 \end{aligned} \tag{B.2}$$

### B.1.3 Add-Instance Rule

The Add-Instance rule adds an Instance matcher to the matching process if the relevance condition evaluates to true (see Figure B.3).

The Instance matcher compares the instances of source and target elements. In its current version it serializes the instance structures into strings and applies a Soft-TFIDF measure to compute the similarity. Since there are currently no other features than Instance-Existence defined to analyze instances of the source and target schemas the following simple condition is used:

$$\begin{aligned} Condition_{Instance} = InstanceExistence(S) > 0 \\ \land InstanceExistence(T) > 0 \end{aligned} \tag{B.3}$$

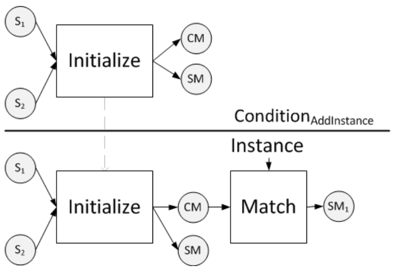


Figure B.3: Add-Instance rule

### B.1.4 Add-Restriction Rule

The Add-Restriction rule as shown in Figure B.4 adds a Match operator to the process that compares restrictions an cardinalities of source and target elements. Possible

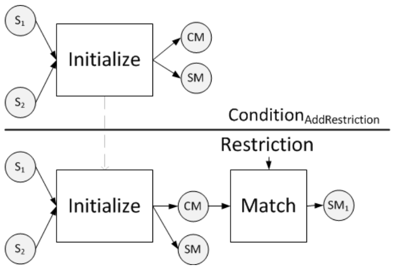


Figure B.4: Add-Restriction rule

cardinalities of each element are collected (an element can be used multiple times and can have different cardinalities depending on the context) and again serialized to strings. These strings are sorted and compared using string similarity techniques. This is a very basic version of a matcher since cardinalities and restrictions could also be compared based on their semantics. The similarity of 1:1 and 1:2 is higher than for instance 3:n and 1:1. Similar to the Add-Instance rule the relevance condition of the Add-Restriction rule is defined solely with the respective Attribute-Existence feature:

$$\begin{aligned} Condition_{Restriction} = {} & RestrictionExistence(S) > 0 \\ & \wedge RestrictionExistence(T) > 0 \end{aligned} \tag{B.4}$$

## B.2 Refine Rules

### B.2.1 Add-Leaf Rule

The Add-Leaf rule adds a Leaf matcher to the process (see Figure B.5).

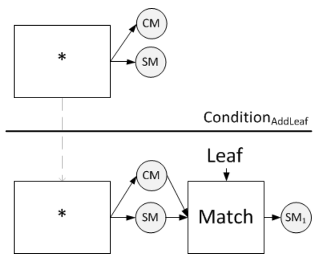


Figure B.5: Add-Leaf rule

The Leaf matcher derives a similarity value for an element pair from comparing the leafs of both elements with each other. The path from an element to its leafs can be long so that a high structural similarity is necessary when a Leaf matcher is used. Therefore it should only be used if the structural similarity of the source and target schema is high. This results in the following simple relevance condition.

$$Condition_{AddLeaf} = StructuralSimilarity(S,T) > 0.5 \tag{B.5}$$

A structural similarity higher $0.5$ gives some indication whether to use the Leaf matcher or not. However, Leaf matcher specific features should be defined in future. For instance, a feature could take the input mapping from a basic matcher as indicator. If there are many non-leaf matches where the leafs of the source and target element also match then applying the Leaf matcher could be recommended.

## B.2.2 Add-Sibling Rule

The Add-Sibling rule is shown in Figure B.6. The Structural-Similarity feature value

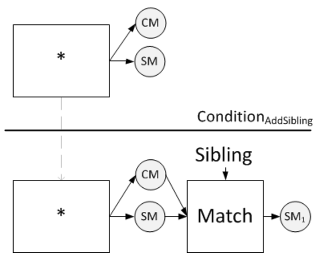


Figure B.6: Add-Sibling rule

of the input schemas should be high when adding the Sibling matcher to the process. Additionally, a Sibling-specific feature is used that measures the distribution of correspondences within siblings. The rational is the following. If two matches point to siblings in the source they should also point to siblings in the target. The more such match pairs an input mapping contains, the more likely it is that applying a Sibling matcher is appropriate. The relevance condition is therefore defined as follows:

$$Condition_{AddSibling} = StructuralSimilarity(S,T) > 0.3 \quad \wedge SiblingDistribution(SM) < 0.3 \tag{B.6}$$

## B.3 Rewrite Rules

### B.3.1 Noise-Filter Rule

If one of the inputs to a Combine operator contains noise ($noise > 0.3$), then the Noise-Filter rule can be applied. It adds a Filter operator between the input operator that produces noise and the Combine operator. The filter threshold is computed similar to the Noise feature and the lower bound is taken as threshold.

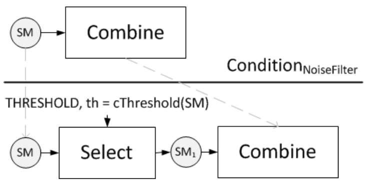


Figure B.7: Noise-Filter Rule

Similar to the Noise feature a histogram $h(SM)$ of input values that consists of $B$ histogram buckets $\{h(SM)_1, \ldots, h(SM)_B\}$ is created. The b-th bucket spans an interval and it is filled with element pairs as described above. All bucket sizes starting with the bucket $b = 1$ are summed up until 20% of all pairs are collected. The index of the current bucket is kept as $low$.

$$low = \{b \in B \mid \sum_x^b |h(SM)_x| > 0.2 \cdot |S \times T| \wedge \sum_x^{b-1} |h(SM)_x| < 0.2 \cdot |S \times T|\} \tag{B.7}$$

The threshold that is taken for the Select operator is then computed as follows:

$$cThreshold = \frac{low}{B} \tag{B.8}$$

### B.3.2 Blocking-Filter Rule

The Blocking-Filter rule is shown in Figure B.8.

To compute the relevance, the following steps are performed. All input similarity matrices $SM_1, \ldots, SM_N$ without the Type matcher result $SM_{dt}$ are combined. Then,

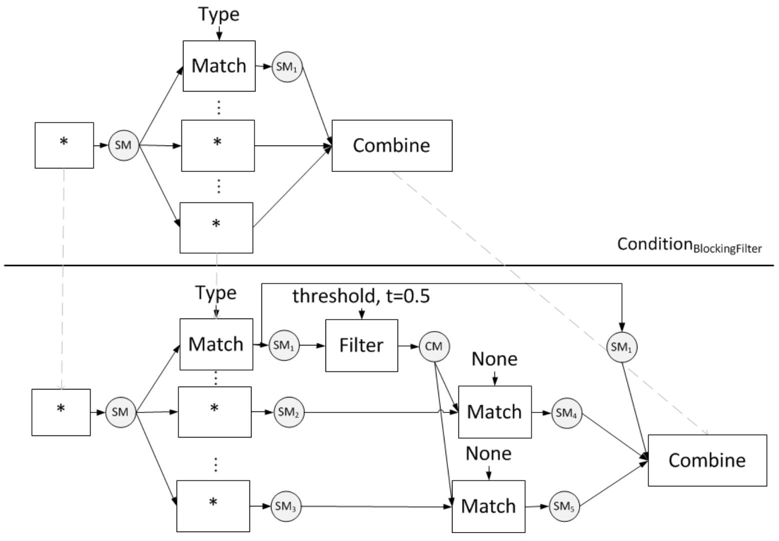


Figure B.8: Blocking-Filter rule

the combined result is selected with a very restrictive set of selections to compute a high precision result $highPres$.

$$\begin{aligned} comb &= Combine_{average}(SM_1, \ldots, SM_N/SM_{dt} \\ highPres &= Select_{maxN,exact,threshold}(comb, N=1, th=0.8) \end{aligned} \tag{B.9}$$

From this high precision result the number of false negatives is counted by comparing the Type matcher similarity values $sm_{i,j}^{dt}$ with the high precision matrix.

$$countNegative = \{p_{i,j} \in highPres \,|\, p_{i,j} > 0 \wedge sm_{i,j}^{dt} < 0.5\} \tag{B.10}$$

If there are many pairs in the high precision matrix with similarity value $> 0$ that are low-valued in the Type matcher result, then the type blocking should not be done. The resulting relevance condition is defined as follows:

$$Condition_{BlockingFilter} = \frac{countNegative}{Min(|S|, |T|)} < 0.01 \tag{B.11}$$

## B.4 Selection Rules

### B.4.1 Max1-Select Rule

In many mapping scenarios only the top 1 result for a selection is needed. In particular if schema elements and fragments do not repeat the top 1 result should contain the correct match. The Select-Max1 rule adds a single Select operator to the process as visualized in Figure B.9.

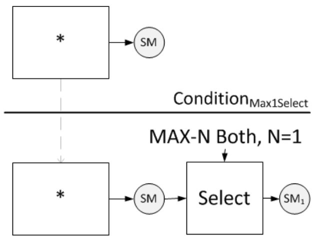


Figure B.9: Max1-Select rule

It could be observed that for many mapping problems it is possible to measure with matrix and schema features what selection operator should be chosen. If the selected mapping has a Match-Count-Ratio close to 1 and elements are not repeating in the source and target schema then a MAX-N selection with N=1 is beneficial. The respective condition can be written as follows:

$$\begin{aligned} Condition_{Max1Select} = \\ MatchCountRatio(M) > 0.85 \wedge AVG(RepeatingElements, S, T) > 0.95 \end{aligned} \tag{B.12}$$

## B.4.2 Adaptive-Threshold Rule

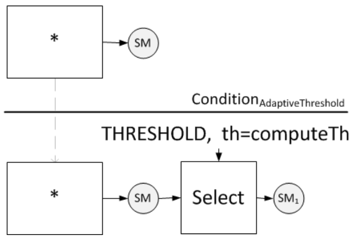


Figure B.10: Adaptive-Threshold rule

The Adaptive-Threshold rule adds a Select operator with selection strategy THRESHOLD to the most recent selection (see Figure B.9). The relevance condition always computes to true. A problem in many matching systems is the definition of appropriate thresholds. In this thesis, the Monogamy value is used to compute the threshold from the input mapping by testing different thresholds and picking the threshold that resulted in highest Monogamy. The threshold that results in highest Monogamy is then taken as threshold for the Select operator.

$$mvalue(SM, x) = Monogamy\left(Select_{Threshold}\left(Select_{Delta}\left(SM, d = 0.01\right), th = x\right)\right) \tag{B.13}$$

$$computeTh(SM) = t \in [0,1] \,|\, mvalue(SM,t) \geq mvalue(SM,y), \forall y \in [0,1] \quad \text{(B.14)}$$

### B.4.3 Restrict-to-N:N

Many mapping problems contain schemas with repeating substructures. In these cases it is more probable that the correct match results contain multi-mappings. However, the cardinality of multi-mappings still needs to be restricted. For that purpose the Restrict-to-N:N rule restricts the cardinality of a mapping by using an N-N selection.

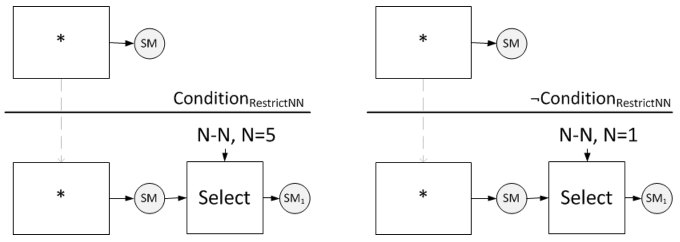


Figure B.11: Restrict-to-N:N rule

Also, a select operator with 1:1 restriction could be added (see Figure B.11). The selection removes the remaining multi-matches from the result to be consistent with the other 1:1 matches. If there are some multi-mappings contained in a result and the Match-Count-Ratio is low, meaning that the majority of computed matches are not 1:1 matches. The relevance condition is formulated accordingly:

$$\begin{aligned} Condition_{RestrictNN} = {} & MultiMappings(SM) < 0.9 \wedge \\ & MatchCountRatio(M) < 0.9 \end{aligned} \quad \text{(B.15)}$$

# Appendix C

# Evaluation Data

## C.1 Data Set

| Scenario | # Source Elements | # Target Elements | # Correspondences | AnnotatoinExistence(S) | AnnotationExistence(T) | NameVariance(S) | NameVariance(T) | ElementTokenRatio(S) | ElementTokenRatio(T) | NameSimilarity | StringTokenOverlap | StructuralSimilarity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| O_101 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 1,00 |
| O_103 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 0,67 |
| O_104 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 0,83 |
| O_201-2 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,90 | 0,98 | 0,68 | 0,75 |
| O_201-4 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,94 | 0,76 | 0,48 | 0,83 |
| O_201-6 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,97 | 0,54 | 0,33 | 0,83 |
| O_201-8 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,99 | 0,29 | 0,19 | 0,83 |
| O_201 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 1,00 | 0,07 | 0,07 | 0,80 |
| O_202-2 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,93 | 0,94 | 0,64 | 0,91 |
| O_202-4 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,71 | 0,44 | 0,78 |
| O_202-6 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,47 | 0,29 | 0,87 |
| O_202-8 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,23 | 0,15 | 0,83 |
| O_202 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,01 | 0,88 |
| O_203 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,89 | 1,13 | 0,98 | 0,76 |
| O_204 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 0,86 | 0,74 | 0,91 |
| O_205 | 97 | 96 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,93 | 0,28 | 0,24 | 0,77 |
| O_206 | 97 | 96 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,90 | 0,31 | 0,19 | 0,83 |
| O_207 | 97 | 96 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,90 | 0,31 | 0,19 | 0,87 |
| O_208 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,88 | 0,81 | 0,72 | 0,86 |
| O_209 | 97 | 96 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,93 | 0,23 | 0,20 | 0,82 |
| O_210 | 97 | 96 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,88 | 0,25 | 0,13 | 0,83 |
| O_221 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 0,50 |
| O_222 | 97 | 93 | 93 | 0,92 | 0,91 | 1,00 | 1,00 | 0,87 | 0,87 | 1,11 | 0,98 | 0,77 |
| O_223 | 97 | 131 | 97 | 0,92 | 0,91 | 1,00 | 0,99 | 0,87 | 0,86 | 0,85 | 0,69 | 0,27 |
| O_224 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 0,86 |
| O_225 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 0,82 |
| O_228 | 97 | 33 | 33 | 0,92 | 1,00 | 1,00 | 1,00 | 0,87 | 0,92 | 0,42 | 0,43 | 0,36 |
| O_230 | 97 | 77 | 72 | 0,92 | 0,90 | 1,00 | 1,00 | 0,87 | 0,92 | 0,80 | 0,83 | 0,68 |
| O_231 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 0,87 |
| O_232 | 97 | 97 | 97 | 0,92 | 0,92 | 1,00 | 1,00 | 0,87 | 0,87 | 1,16 | 1,00 | 0,45 |
| O_233 | 97 | 33 | 33 | 0,92 | 1,00 | 1,00 | 1,00 | 0,87 | 0,92 | 0,42 | 0,43 | 0,00 |
| O_236 | 97 | 33 | 33 | 0,92 | 1,00 | 1,00 | 1,00 | 0,87 | 0,92 | 0,42 | 0,43 | 0,36 |
| O_237 | 97 | 93 | 93 | 0,92 | 0,91 | 1,00 | 1,00 | 0,87 | 0,87 | 1,11 | 0,98 | 0,77 |
| O_238 | 97 | 131 | 97 | 0,92 | 0,91 | 1,00 | 0,99 | 0,87 | 0,86 | 0,85 | 0,69 | 0,33 |
| O_239 | 97 | 30 | 29 | 0,92 | 0,97 | 1,00 | 1,00 | 0,87 | 0,94 | 0,38 | 0,43 | 0,09 |
| O_240 | 97 | 67 | 33 | 0,92 | 0,93 | 1,00 | 0,99 | 0,87 | 0,88 | 0,42 | 0,33 | 0,32 |
| O_241 | 97 | 33 | 33 | 0,92 | 1,00 | 1,00 | 1,00 | 0,87 | 0,92 | 0,42 | 0,43 | 0,00 |
| O_246 | 97 | 30 | 29 | 0,92 | 0,97 | 1,00 | 1,00 | 0,87 | 0,94 | 0,38 | 0,43 | 0,09 |
| O_247 | 97 | 67 | 33 | 0,92 | 0,93 | 1,00 | 0,99 | 0,87 | 0,88 | 0,42 | 0,33 | 0,32 |
| O_248-2 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,93 | 0,94 | 0,64 | 0,45 |
| O_248-4 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,71 | 0,44 | 0,45 |
| O_248-6 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,47 | 0,29 | 0,45 |
| O_248-8 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,23 | 0,15 | 0,45 |
| O_248 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,01 | 0,50 |
| O_249-2 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,93 | 0,94 | 0,64 | 0,87 |
| O_249-4 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,71 | 0,44 | 0,87 |
| O_249-6 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,47 | 0,29 | 0,87 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| O_249-8 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,23 | 0,15 | 0,75 |
| O_249 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,01 | 0,78 |
| O_250-2 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,34 | 0,29 | 0,36 |
| O_250-4 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,26 | 0,22 | 0,36 |
| O_250-6 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,18 | 0,16 | 0,36 |
| O_250-8 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,07 | 0,08 | 0,36 |
| O_250 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,00 | 0,00 | 0,36 |
| O_251-2 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,93 | 0,90 | 0,64 | 0,73 |
| O_251-4 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,68 | 0,44 | 0,77 |
| O_251-6 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,44 | 0,29 | 0,73 |
| O_251-8 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,23 | 0,15 | 0,73 |
| O_251 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,01 | 0,77 |
| O_252-2 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,27 |
| O_252-4 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,27 |
| O_252-6 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,33 |
| O_252-8 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,33 |
| O_252 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,01 | 0,01 | 0,32 |
| O_253-2 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,93 | 0,94 | 0,64 | 0,45 |
| O_253-4 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,71 | 0,44 | 0,45 |
| O_253-6 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,47 | 0,29 | 0,43 |
| O_253-8 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,23 | 0,15 | 0,50 |
| O_253 | 97 | 97 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,01 | 0,45 |
| O_254-2 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,34 | 0,29 | 0,00 |
| O_254-4 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,26 | 0,22 | 0,00 |
| O_254-6 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,18 | 0,16 | 0,00 |
| O_254-8 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,07 | 0,08 | 0,00 |
| O_254 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,00 | 0,00 | 0,00 |
| O_257-2 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,34 | 0,29 | 0,36 |
| O_257-4 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,26 | 0,22 | 0,36 |
| O_257-6 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,18 | 0,16 | 0,36 |
| O_257-8 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,07 | 0,08 | 0,36 |
| O_257 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,00 | 0,00 | 0,36 |
| O_258-2 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,93 | 0,90 | 0,64 | 0,73 |
| O_258-4 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,68 | 0,44 | 0,73 |
| O_258-6 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,44 | 0,29 | 0,77 |
| O_258-8 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,23 | 0,15 | 0,82 |
| O_258 | 97 | 93 | 93 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,01 | 0,73 |
| O_259-2 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,33 |
| O_259-4 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,26 |
| O_259-6 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,27 |
| O_259-8 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,94 | 0,69 | 0,46 | 0,33 |
| O_259 | 97 | 131 | 97 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,01 | 0,01 | 0,34 |
| O_260-2 | 97 | 30 | 29 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,97 | 0,31 | 0,29 | 0,09 |
| O_260-4 | 97 | 30 | 29 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,24 | 0,23 | 0,09 |
| O_260-6 | 97 | 30 | 29 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,15 | 0,16 | 0,09 |
| O_260-8 | 97 | 30 | 29 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,08 | 0,10 | 0,09 |
| O_260 | 97 | 30 | 29 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,02 | 0,09 |
| O_261-2 | 97 | 67 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,95 | 0,35 | 0,21 | 0,32 |
| O_261-4 | 97 | 67 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,95 | 0,35 | 0,21 | 0,32 |
| O_261-6 | 97 | 67 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,95 | 0,35 | 0,21 | 0,32 |
| O_261-8 | 97 | 67 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,95 | 0,35 | 0,21 | 0,32 |
| O_261 | 97 | 67 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,98 | 0,01 | 0,01 | 0,32 |
| O_262-2 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,96 | 0,34 | 0,29 | 0,00 |
| O_262-4 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,99 | 0,26 | 0,22 | 0,00 |
| O_262-6 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,18 | 0,16 | 0,00 |
| O_262-8 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,07 | 0,08 | 0,00 |
| O_262 | 97 | 33 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,00 | 0,00 | 0,00 |
| O_265 | 97 | 30 | 29 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 1,00 | 0,01 | 0,02 | 0,09 |
| O_266 | 97 | 67 | 33 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,98 | 0,01 | 0,01 | 0,32 |
| O_301 | 97 | 55 | 59 | 0,92 | 1,00 | 1,00 | 1,00 | 0,87 | 0,98 | 0,18 | 0,48 | 0,14 |
| O_302 | 97 | 43 | 48 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,91 | 0,35 | 0,30 | 0,18 |
| O_303 | 97 | 126 | 48 | 0,92 | 0,00 | 1,00 | 1,00 | 0,87 | 0,85 | 0,39 | 0,28 | 0,39 |
| O_304 | 97 | 88 | 76 | 0,92 | 0,94 | 1,00 | 0,99 | 0,87 | 0,84 | 0,92 | 0,74 | 0,74 |
| PO_1 | 40 | 147 | 54 | 0,00 | 0,00 | 0,91 | 0,84 | 0,77 | 0,57 | 0,14 | 0,19 | 0,13 |
| PO_2 | 40 | 54 | 41 | 0,00 | 0,00 | 0,91 | 0,87 | 0,77 | 0,65 | 0,74 | 0,40 | 0,23 |
| PO_3 | 40 | 65 | 36 | 0,00 | 0,00 | 0,91 | 0,90 | 0,77 | 0,67 | 0,22 | 0,19 | 0,33 |
| PO_4 | 40 | 80 | 49 | 0,00 | 0,01 | 0,91 | 0,91 | 0,77 | 0,66 | 0,29 | 0,25 | 0,31 |
| PO_5 | 54 | 147 | 79 | 0,00 | 0,00 | 0,87 | 0,84 | 0,65 | 0,57 | 0,28 | 0,23 | 0,37 |
| PO_6 | 54 | 65 | 50 | 0,00 | 0,00 | 0,87 | 0,90 | 0,65 | 0,67 | 0,34 | 0,27 | 0,54 |
| PO_7 | 54 | 80 | 60 | 0,00 | 0,01 | 0,87 | 0,91 | 0,65 | 0,66 | 0,45 | 0,17 | 0,31 |
| PO_8 | 65 | 147 | 85 | 0,00 | 0,00 | 0,90 | 0,84 | 0,67 | 0,57 | 0,47 | 0,35 | 0,30 |
| PO_9 | 65 | 80 | 45 | 0,00 | 0,01 | 0,90 | 0,91 | 0,67 | 0,66 | 0,41 | 0,19 | 0,69 |
| PO_10 | 80 | 147 | 66 | 0,01 | 0,00 | 0,91 | 0,84 | 0,66 | 0,57 | 0,49 | 0,22 | 0,17 |
| ES_1 | 49 | 7 | 6 | 0,90 | 1,00 | 0,99 | 1,00 | 0,99 | 0,80 | 0,02 | 0,02 | 0,20 |
| ES_2 | 1042 | 1042 | 1042 | 0,93 | 0,93 | 0,92 | 0,92 | 0,81 | 0,81 | 5,60 | 1,00 | 1,00 |
| ES_3 | 1164 | 1164 | 1164 | 0,93 | 0,93 | 0,91 | 0,91 | 0,80 | 0,80 | 7,01 | 1,00 | 1,00 |
| ES_4 | 1164 | 1164 | 1164 | 0,93 | 0,93 | 0,91 | 0,91 | 0,80 | 0,80 | 7,01 | 1,00 | 1,00 |
| ES_5 | 1307 | 1307 | 1307 | 0,93 | 0,93 | 0,91 | 0,91 | 0,78 | 0,78 | 8,06 | 1,00 | 1,00 |
| ES_6 | 8 | 71 | 7 | 1,00 | 0,90 | 0,92 | 0,98 | 0,86 | 0,91 | 0,00 | 0,09 | 0,14 |
| ES_7 | 8 | 66 | 6 | 1,00 | 0,89 | 0,92 | 0,98 | 0,86 | 0,92 | 0,00 | 0,09 | 0,14 |
| ES_8 | 11 | 9 | 9 | 1,00 | 1,00 | 1,00 | 1,00 | 0,83 | 0,84 | 0,82 | 0,89 | 0,80 |
| ES_9 | 145 | 137 | 74 | 0,99 | 0,00 | 0,94 | 0,96 | 0,68 | 0,68 | 0,09 | 0,44 | 0,27 |
| ES_10 | 70 | 14 | 13 | 0,97 | 1,00 | 0,87 | 1,00 | 0,61 | 0,89 | 0,19 | 0,35 | 0,12 |
| ES_11 | 167 | 131 | 67 | 0,00 | 0,98 | 0,90 | 0,93 | 0,67 | 0,70 | 0,07 | 0,47 | 0,37 |
| ES_12 | 46 | 66 | 25 | 0,98 | 0,32 | 0,99 | 0,98 | 0,91 | 0,79 | 0,17 | 0,06 | 0,16 |
| ES_13 | 17 | 13 | 13 | 0,71 | 1,00 | 1,00 | 1,00 | 0,94 | 0,96 | 0,41 | 0,35 | 0,13 |
| ES_14 | 70 | 101 | 73 | 0,99 | 0,65 | 1,00 | 0,89 | 0,97 | 0,89 | 0,14 | 0,13 | 0,03 |
| ES_15 | 36 | 51 | 30 | 0,97 | 0,71 | 1,00 | 0,93 | 0,95 | 0,88 | 0,18 | 0,16 | 0,05 |
| ES_16 | 44 | 63 | 40 | 0,98 | 0,73 | 1,00 | 0,94 | 0,96 | 0,90 | 0,14 | 0,10 | 0,09 |
| ES_17 | 31 | 43 | 28 | 1,00 | 0,70 | 0,99 | 1,00 | 0,95 | 0,95 | 0,12 | 0,12 | 0,13 |
| ES_18 | 34 | 45 | 30 | 1,00 | 0,71 | 0,99 | 1,00 | 0,95 | 0,95 | 0,16 | 0,14 | 0,13 |
| ES_19 | 29 | 42 | 26 | 1,00 | 0,69 | 0,99 | 0,99 | 0,94 | 0,93 | 0,12 | 0,13 | 0,12 |
| ES_20 | 51 | 83 | 62 | 1,00 | 0,65 | 0,99 | 0,90 | 0,97 | 0,84 | 0,10 | 0,10 | 0,03 |
| ES_21 | 136 | 197 | 144 | 0,98 | 0,73 | 1,00 | 0,83 | 0,91 | 0,81 | 0,23 | 0,13 | 0,05 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ES_22 | 19 | 35 | 19 | 1,00 | 0,63 | 0,98 | 0,96 | 0,92 | 0,88 | 0,14 | 0,16 | 0,06 |
| ES_23 | 55 | 249 | 15 | 0,96 | 1,00 | 0,83 | 0,80 | 0,63 | 0,43 | 0,48 | 0,27 | 0,08 |
| ES_24 | 54 | 249 | 15 | 0,96 | 1,00 | 0,83 | 0,80 | 0,63 | 0,43 | 0,48 | 0,27 | 0,08 |
| ES_25 | 24 | 41 | 12 | 1,00 | 1,00 | 0,86 | 0,82 | 0,73 | 0,46 | 0,00 | 0,17 | 0,36 |
| ES_26 | 30 | 16 | 16 | 0,57 | 0,75 | 0,93 | 1,00 | 0,74 | 0,92 | 0,70 | 0,83 | 0,54 |
| ES_27 | 66 | 52 | 51 | 0,42 | 0,38 | 0,87 | 0,88 | 0,68 | 0,75 | 1,76 | 0,90 | 0,85 |
| ES_28 | 55 | 43 | 44 | 0,31 | 0,28 | 0,90 | 0,90 | 0,72 | 0,81 | 1,62 | 0,89 | 0,87 |
| ES_29 | 75 | 6 | 5 | 0,28 | 0,67 | 0,76 | 1,00 | 0,56 | 0,83 | 0,11 | 0,25 | 0,03 |
| ES_30 | 509 | 13 | 10 | 0,87 | 0,62 | 0,86 | 1,00 | 0,72 | 0,71 | 0,02 | 0,04 | 0,01 |
| ES_31 | 64 | 46 | 49 | 0,33 | 0,26 | 0,89 | 0,91 | 0,72 | 0,84 | 1,56 | 0,89 | 0,75 |
| ES_32 | 52 | 7 | 4 | 0,27 | 0,00 | 0,92 | 1,00 | 0,81 | 0,81 | 0,10 | 0,17 | 0,20 |
| ES_33 | 6 | 5 | 4 | 0,00 | 0,00 | 1,00 | 1,00 | 0,92 | 0,75 | 0,33 | 0,33 | 0,33 |
| ES_34 | 60 | 48 | 49 | 0,37 | 0,35 | 0,91 | 0,91 | 0,75 | 0,82 | 1,57 | 0,91 | 0,87 |
| ES_35 | 29 | 36 | 18 | 0,00 | 0,94 | 0,97 | 0,86 | 0,69 | 0,52 | 0,14 | 0,28 | 0,25 |
| ES_36 | 181 | 24 | 23 | 0,01 | 1,00 | 0,99 | 0,85 | 0,78 | 0,55 | 0,02 | 0,07 | 0,30 |
| ES_37 | 29 | 36 | 18 | 0,00 | 0,94 | 0,97 | 0,86 | 0,69 | 0,52 | 0,14 | 0,28 | 0,25 |
| ES_38 | 163 | 24 | 23 | 0,00 | 1,00 | 0,99 | 0,85 | 0,79 | 0,55 | 0,02 | 0,08 | 0,24 |
| ES_39 | 18 | 16 | 18 | 1,00 | 0,00 | 1,00 | 1,00 | 0,76 | 0,76 | 0,56 | 0,82 | 0,33 |
| ES_40 | 12 | 14 | 9 | 1,00 | 1,00 | 1,00 | 1,00 | 0,89 | 0,82 | 0,43 | 0,68 | 0,75 |
| ES_41 | 12 | 59 | 12 | 1,00 | 0,90 | 1,00 | 0,98 | 0,89 | 0,97 | 0,03 | 0,01 | 0,17 |
| ES_42 | 14 | 59 | 14 | 1,00 | 0,90 | 1,00 | 0,98 | 0,82 | 0,97 | 0,05 | 0,01 | 0,33 |
| ES_43 | 10 | 7 | 6 | 1,00 | 1,00 | 1,00 | 1,00 | 0,73 | 0,70 | 0,60 | 0,67 | 0,50 |
| ES_44 | 54 | 7 | 6 | 0,87 | 1,00 | 0,97 | 1,00 | 0,96 | 0,70 | 0,04 | 0,03 | 0,14 |
| ES_45 | 54 | 10 | 8 | 0,87 | 1,00 | 0,97 | 1,00 | 0,96 | 0,73 | 0,06 | 0,03 | 0,14 |
| ES_46 | 12 | 11 | 6 | 0,83 | 0,82 | 1,00 | 1,00 | 0,77 | 0,79 | 0,75 | 0,69 | 0,67 |
| ES_47 | 27 | 50 | 36 | 0,93 | 0,94 | 1,00 | 0,89 | 0,84 | 0,53 | 0,82 | 0,85 | 0,56 |
| ES_48 | 13 | 65 | 7 | 1,00 | 0,88 | 0,88 | 0,92 | 0,79 | 0,83 | 0,14 | 0,11 | 0,07 |
| ES_49 | 53 | 27 | 14 | 0,89 | 1,00 | 0,93 | 1,00 | 0,87 | 0,69 | 0,13 | 0,15 | 0,17 |
| EDI_1 | 130 | 88 | 40 | 0,99 | 0,99 | 0,96 | 0,99 | 0,73 | 0,86 | 0,44 | 0,41 | 0,72 |
| EDI_2 | 121 | 240 | 17 | 0,99 | 1,00 | 0,84 | 0,88 | 0,58 | 0,46 | 0,02 | 0,16 | 0,44 |
| EDI_3 | 121 | 73 | 16 | 0,99 | 1,00 | 0,84 | 0,97 | 0,58 | 0,71 | 0,00 | 0,00 | 0,21 |
| EDI_4 | 121 | 285 | 21 | 0,99 | 0,39 | 0,84 | 0,65 | 0,58 | 0,60 | 0,00 | 0,26 | 0,42 |
| EDI_5 | 73 | 53 | 4 | 1,00 | 0,92 | 0,97 | 0,75 | 0,71 | 0,62 | 0,00 | 0,00 | 0,56 |
| EDI_6 | 240 | 360 | 81 | 1,00 | 1,00 | 0,88 | 0,88 | 0,46 | 0,44 | 1,21 | 0,44 | 0,54 |
| ANA_1 | 2739 | 3299 | 1520 | 0,00 | 0,00 | 1,00 | 1,00 | 0,59 | 0,66 | 0,06 | 0,11 | 0,35 |

## C.2 Incidence Graph from Evaluation

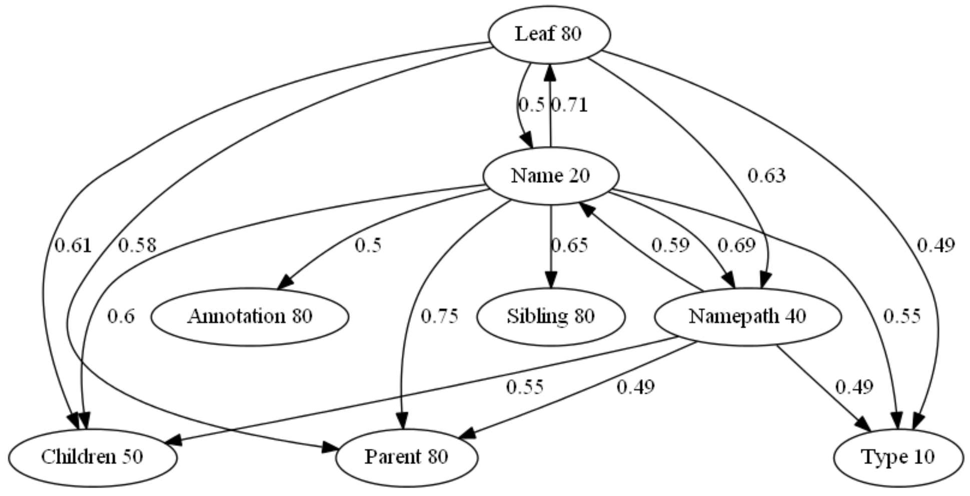


Figure C.1: Incidence graph used within evaluation

## C.3 Rewritten Matching Processes

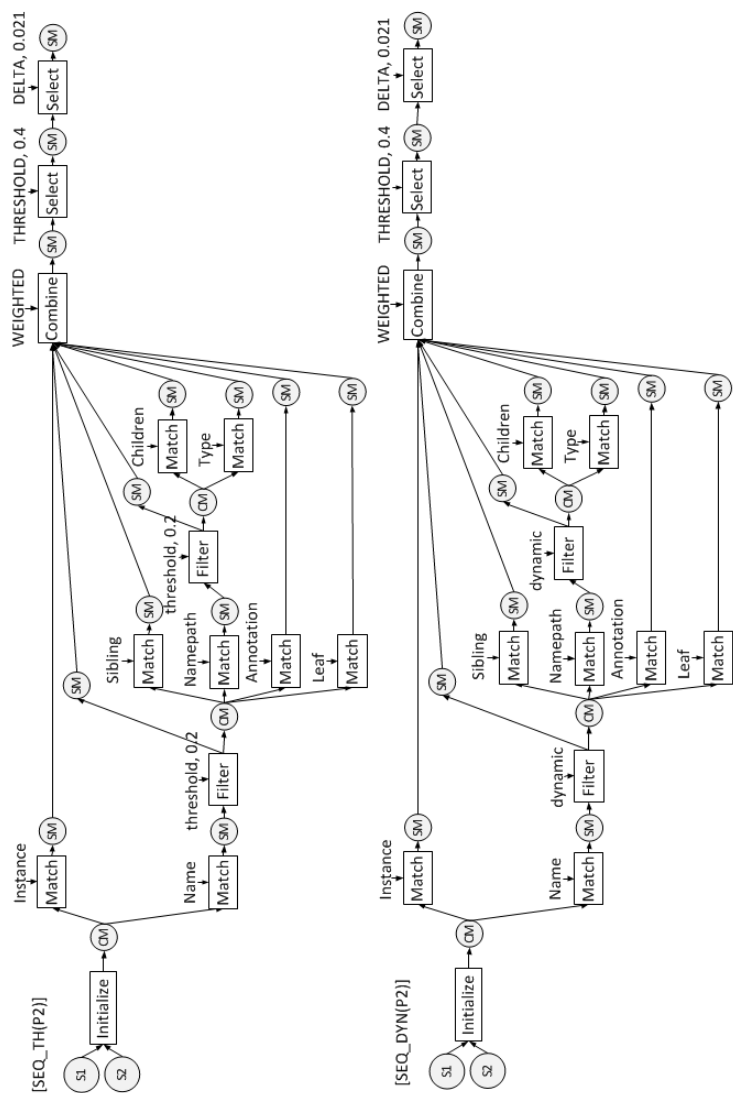


Figure C.2: Rewritten matching processes SEQ-TH and SEQ-DYN for P2

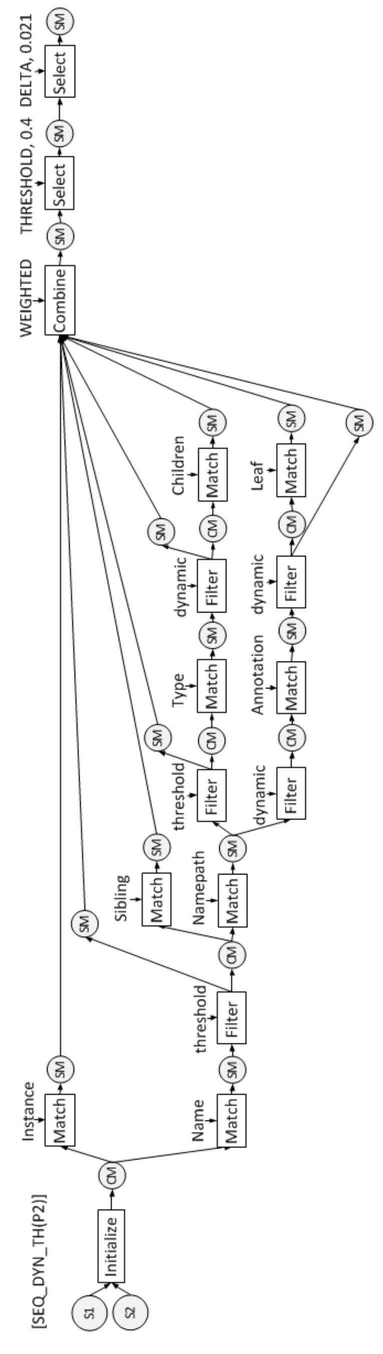


Figure C.3: Rewritten matching process SEQ-DYN-TH for P2

# Bibliography

[1] UN/CEFACT: Core Component Technical Specification V2.01, 2006. http://www.unece.org/ [last visited may 19th 2013].

[2] S K Srivatsa A Rajesh. Learning to Match XML Schemas: A Decision Tree based Approach. *International Journal of Recent Trends in Engineering*, (2), 2009.

[3] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *VLDB Proceedings*, pages 411–422, 2007.

[4] Karl Aberer, Philippe Cudré-Mauroux, and Manfred Hauswirth. A Framework for Semantic Gossiping. *SIGMOD Record*, 31(4):48–53, 2002.

[5] Rakesh Agrawal and Ramakrishnan Srikant. On Integrating Catalogs. In *WWW Proceedings*, pages 603–612, 2001.

[6] Bogdan Alexe, Laura Chiticariu, Renée J. Miller, and Wang Chiew Tan. Muse: Mapping Understanding and deSign by Example. In *ICDE Proceedings*, pages 10–19, 2008.

[7] Alsayed Algergawy. *Management of XML Data by Means of Schema Matching*. Dissertation, University of Magdeburg, Germany, February 2010.

[8] Alsayed Algergawy, Eike Schallehn, and Gunter Saake. A New XML Schema Matching Approach Using Prüfer Sequences. In *DB&IS*, pages 217–228, 2008.

[9] Altova Mapforce. http://www.altova.com/de/mapforce.html [last visited may 19th 2013].

[10] M. Ashburner. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[11] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and Ontology Matching with COMA++. In *SIGMOD Proceedings*, pages 906–908, 2005.

[12] Paolo Avesani, Fausto Giunchiglia, and Mikalai Yatskevich. A Large Scale Taxonomy Mapping Evaluation. In *ISWC Proceedings*, volume 3729, pages 67–81, 2005.

[13] Babak Bagheri Hariri, Hassan Sayyadi, Hassan Abolhassani, and Kyumars Sheykh Esmaili. Combining Ontology Alignment Metrics Using the Data Mining Techniques. In *17th European Conference on Artificial Intelligence, International Workshop on Context and Ontologies Representation and Reasoning (C&O'06)*, 2006.

[14] A. Baroni, S. Braz, and F. Abreu. Using OCL to Formalize Object-Oriented Design Metrics Definitions. In *ECOOP'02 Workshop on Quantitative Approaches in OO Software Engineering*, 2002.

[15] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[16] Rohan Baxter, Peter Christen, and Tim Churches. A Comparison of Fast Blocking Methods for Record Linkage. In *ACM SIGKDD '03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27, 2003.

[17] Khalid Belhajjame, Norman W. Paton, Alvaro A. A. Fernandes, Cornelia Hedeler, and Suzanne M. Embury. User Feedback as a First Class Citizen in Information Integration Systems. In *CIDR Proceedings Fifth Biennial Conference on Innovative Data Systems Research*, pages 175–183, 2011.

[18] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm, editors. *Schema Matching and Mapping*. Springer, 2011.

[19] Shlomo Berkovsky, Yaniv Eytani, and Avigdor Gal. Measuring the Relative Performance of Schema Matchers. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 366–371, 2005.

[20] Jacob Berlin and Amihai Motro. Database Schema Matching Using Machine Learning with Feature Selection. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, pages 452–466, 2002.

[21] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[22] Philip A. Bernstein and Sergey Melnik. Model Management 2.0: Manipulating Richer Mappings. In *SIGMOD Proceedings*, pages 1–12, 2007.

[23] Philip A. Bernstein, Sergey Melnik, and John E. Churchills. Incremental Schema Matching. *In VLDB Proceedings*, pages 1167–1170, 2006.

[24] Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, and Christoph Quix. Industrial-Strength Schema Matching. *SIGMOD Record*, 33(4):38–43, 2004.

[25] Jürgen Bock, Alexander Lenk, and Carsten Dänschel. Ontology Alignment in the Cloud. In *Proceedings of the 5th International Workshop on Ontology Matching (OM-2010)*, volume 689, pages 73–84, 2010.

[26] Angela Bonifati, Giansalvatore Mecca, Alessandro Pappalardo, Salvatore Raunich, and Gianvito Summa. The Spicy System: Towards a Notion of Mapping Quality. In *SIGMOD Proceedings*, pages 1289–1294, 2008.

[27] Jon Bosak, Tim McGrath, and G. Ken Holman. *Universal Business Language v2.0*. December 2006.

[28] Watson Wei Khong Chua and Jung-Jae Kim. Discovering Cross-Ontology Subsumption Relationships by Using Ontological Annotations on Biomedical Literature. In *ICBO Proceedings*, 2012.

[29] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *IIWeb'03 Proceedings*, pages 73–78, 2003.

[30] Isabel Cruz, Alessio Fabiani, Federico Caimi, Cosmin Stroe, and Matteo Palmonari. Automatic Configuration Selection Using Ontology Matching Task Profiling. *In ESWC Proceedings*, 7295:179–194, 2012.

[31] Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies . *In VLDB Proceedings*, 2(2):1586–1589, 2009.

[32] Isabel F. Cruz, Cosmin Stroe, and Matteo Palmonari. Interactive User Feedback in Ontology Matching Using Signature Vectors. In *ICDE Proceedings*, pages 1321–1324, 2012.

[33] Juan de Lara and Gabriele Taentzer. Automated Model Transformation and Its Validation Using AToM 3 and AGG. pages 182–198. 2004.

[34] 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*, pages 963–970, 2007.

[35] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB Proceedings*, pages 610–621, 2002.

[36] Hong-Hai Do. *Schema Matching and Mapping Based Data Integration*. PhD thesis, University of Leipzig, 2005.

[37] Hong Hai Do and Erhard Rahm. Matching Large Schemas: Approaches and Evaluation. *Inf. Syst.*, 32(6):857–885, 2007.

[38] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to Map between Ontologies on the Semantic Web. *In WWW Proceedings*, pages 662–673, 2002.

[39] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. *SIGMOD Record*, 30:509–520, 2001.

[40] Eduard C. Dragut, Wensheng Wu, A. Prasad Sistla, Clement T. Yu, and Weiyi Meng. Merging Source Query Interfaces on Web Databases. In *ICDE Proceedings*, page 46, 2006.

[41] Christian Drumm, Matthias Schmitt, Hong-Hai Do, and Erhard Rahm. QuickMig - Automatic Schema Matching for Data Migration Projects. In *CIKM Proceedings*, 2007.

[42] Songyun Duan, Achille Fokoue, and Kavitha Srinivas. One Size Does Not Fit All: Customizing Ontology Alignment Using User Feedback. In *ISWC Proceedings*, pages 177–192, 2010.

[43] Fabien Duchateau, Zohra Bellahsene, and Remi Coletta. A Flexible Approach for Planning Schema Matching Algorithms. In *OTM '08: Proceedings On the Move to Meaningful Internet Systems*, pages 249–264, 2008.

[44] Fabien Duchateau, Zohra Bellahsene, Mark Roantree, and Mathieu Roche. An Indexing Structure for Automatic Schema Matching. In *ICDE Workshops Proceedings*, pages 485–491, 2007.

[45] Fabien Duchateau, Zohra Bellahsene, and Mathieu Roche. BMatch: a Semantically Context-based Tool Enhanced by an Indexing Structure to Accelerate Schema Matching. In *23èmes Journées Bases de Données Avancées, BDA, Marseille*, 2007.

[46] Fabien Duchateau, Remi Coletta, Zohra Bellahsene, and Renée J. Miller. YAM: a Schema Matcher Factory (Demo). In *CIKM Proceedings*, pages 2079–2080, 2009.

[47] Ted Dunning. *Statistical Identification of Language*. Computing Research Laboratory, New Mexico State University, 1994.

[48] Julian Eberius. Developing a Learning-Based Method for Automatic Optimization of Schema Matching Processes. Master's thesis, Dresden University of Technology, 2011.

[49] Kai Eckert, Christian Meilicke, and Heiner Stuckenschmidt. Improving Ontology Matching Using Meta-level Learning. In *ESWC Proceedings*, pages 158–172, 2009.

[50] Marc Ehrig and Steffen Staab. QOM - Quick Ontology Mapping. In *ISWC Proceedings*, pages 683–697, 2004.

[51] Marc Ehrig, Steffen Staab, and York Sure. Bootstrapping Ontology Alignment Methods with APFEL. In *WWW Proceedings*, pages 1148–1149, 2005.

[52] Marc Ehrig and York Sure. FOAM - Framework for Ontology Alignment and Mapping; Results of the Ontology Alignment Initiative. In *Proceedings of the Workshop on Integrating Ontologies*, volume 156, pages 72–76, 2005.

[53] Jérôme Euzenat et al. Results of the Ontology Alignment Evaluation Initiative 2010. In *OM Proceedings*, 2010.

[54] Jérôme Euzenat et al. Results of the ontology alignment evaluation initiative 2011. *In OM Proceedings*, pages 85–110, 2011.

[55] José Luis Aguirre et al. Results of the ontology alignment evaluation initiative 2012. *In OM Proceedings*, pages 73–115, 2012.

[56] Jérôme Euzenat, Marc Ehrig, Anja Jentzsch, Malgorzata Mochol, and Pavel Shvaiko. Case-Based Recommendation of Matching Tools and Techniques. deliverable 1.2.2.2.1, Knowledge Web Project, 2006.

[57] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer-Verlag, 2007.

[58] Jérôme Euzenat, Heiner Stuckenschmidt, and Mikalai Yatskevich. Introduction to the Ontology Alignment Evaluation 2005. In *Integrating Ontologies*, 2005.

[59] Ronald Fagin, Laura M. Haas, Mauricio Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Conceptual Modeling: Foundations and Applications. chapter Clio: Schema Mapping Creation and Data Exchange, pages 198–236. 2009.

[60] Sean M. Falconer and Margaret-Anne Storey. A Cognitive Support Framework for Ontology Mapping. In *ISWC Proceedings*, pages 114–127, 2007.

[61] Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. Metamodel Matching for Automatic Model Transformation Generation. In *MoDELS '08: Proceedings*, pages 326–340, 2008.

[62] Avigdor Gal, Tomer Sagi, Matthias Weidlich, Eliezer Levy, Victor Shafran, Zoltán Miklós, and Nguyen Quoc Viet Hung. Making Sense of Top-K Matchings. A Unified Match Graph for Schema Matching. In *Proceedings of the Ninth International Workshop on Information Integration on the Web*, pages 6:1–6:6, 2012.

[63] Avigdor Gal and Pavel Shvaiko. Advances in Ontology Matching. In *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*, pages 176–198. 2009.

[64] D. Gale and L. S. Shapley. College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[65] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Anal. Appl.*, 13(1):113–129, 2010.

[66] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-Match: an Algorithm and an Implementation of Semantic Matching. In *Semantic Interoperability and Integration, Dagstuhl Seminar Proceedings*, number 04391, 2005.

[67] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Approximate String Joins in a Database (Almost) for Free. In *VLDB Proceedings*, pages 491–500, 2001.

[68] M. Groß, A.and Hartung, Kirsten T., and E Rahm. GOMMA Results for OAEI 2012. In *Seventh International Workshop on Ontology Matching @ ISWC 2012*, 2012.

[69] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. On Matching Large Life Science Ontologies in Parallel. In *Proceedings of the 7th international conference on Data integration in the life sciences*, pages 35–49, 2010.

[70] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. Mapping Composition for Matching Large Life Science Ontologies. In *ICBO Proceedings*, 2011.

[71] Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[72] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *SIGMOD Proceedings*, pages 805–810, 2005.

[73] Patrick A. V. Hall and Geoff R. Dowling. Approximate String Matching. *ACM Comput. Surv.*, 12:381–402, 1980.

[74] Bin He and Kevin Chen-Chuan Chang. Making Holistic Schema Matching Robust: An Ensemble Approach. In *SIGKDD Proceedings*, pages 429–438, 2005.

[75] Wei Hu, Ningsheng Jian, Yuzhong Qu, and Yanbing Wang. GMO: A Graph Matching for Ontologies. In *Proceedings of Workshop on Integrating Ontologies*, 2005.

[76] Wei Hu and Yuzhong Qu. Block Matching for Ontologies. In *ISWC Proceedings*, pages 300–313, 2006.

[77] Wei Hu and Yuzhong Qu. Falcon-AO: A practical ontology matching system. *Web Semant.*, 6(3):237–239, 2008.

[78] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng.*, 67(1):140–160, 2008.

[79] Wei Hu, Yuanyuan Zhao, Dan Li, Gong Cheng, Honghan Wu, and Yuzhong Qu. Falcon-AO: Results for OAEI 2007. In *Ontology Matching Workshop Proceedings*, 2007.

[80] Mirella Huza, Mounira Harzallah, and Francky Trichet. OntoMas: a Tutoring System Dedicated to Ontology Matching. In *Ontology Matching Workshop Proceedings*, 2006.

[81] Ryutaro Ichise, Masahiro Hamasaki, and Hideaki Takeda. A Multi-strategy Approach for Catalog Integration. In *PRICAI 2004: Trends in Artificial Intelligence*, pages 944–945. 2004.

[82] Antoine Isaac, Lourens Van Der Meij, Stefan Schlobach, and Shenghui Wang. An Empirical Study of Instance-Based Ontology Matching. In *ISWC Proceedings*, 2007.

[83] Till Janner, Fenareti Lampathaki, Volker Hoyer, Spiros Mouzakitis, Yannis Charalabidis, and Christoph Schroth. A Core Component-based Modelling Approach for Achieving e-Business Semantics Interoperability. *J. Theor. Appl. Electron. Commer. Res.*, 3(3):1–16, 2008.

[84] Matthew A. Jaro. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[85] Yves R. Jean-Mary and Mansur R. Kabuka. ASMOV Results for OAEI 2007. In *Ontology Matching Workshop Proceedings*, 2007.

[86] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology Matching with Semantic Verification. *J. Web Sem.*, 7(3):235–251, 2009.

[87] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go User Feedback for Dataspace Systems. In *SIGMOD Proceedings*, pages 847–860, 2008.

[88] Qiu Ji, Peter Haase, and Guilin Qi. Combination of Similarity Measures in Ontology Matching using the OWA Operator. In *Information processing and management of uncertainty in knowledge-based systems (IPMU'08)*, 2008.

[89] Qiu Ji, Weiru Liu, Guilin Qi, and David A. Bell. LCS: A Linguistic Combination System for Ontology Matching. In *KSEM Proceedings*, pages 176–189, 2006.

[90] Gerti Kappel, Horst Kargl, Gerhard Kramler, Andrea Schauerhuber, Martina Seidl, Michael Strommer, and Manuel Wimmer. Matching Metamodels with Semantic Systems - An Experience Report. In *BTW Workshops Proceedings*, pages 38–52, 2007.

[91] Toralf Kirsten, Andreas Thor, and Erhard Rahm. Instance-Based Matching of Large Life Science Ontologies. In *DILS Proceedings*, pages 172–187, 2007.

[92] Alan Kotok and David R. R. Webber. *EbXML: The New Global Standard for Doing Business Over the Internet*. Sams Publishing, 2001.

[93] Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record Linkage: Similarity Measures and Algorithms. In *SIGMOD Proceedings*, pages 802–803, 2006.

[94] Harold W. Kuhn. The Hungarian Method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[95] Monika Lanzenberger and Jennifer Sampson. AlViz - A Tool for Visual Ontology Alignment. In *Proceedings of the conference on Information Visualization*, pages 430–440, 2006.

[96] J. A. Larson, S. B. Navathe, and R. Elmasri. A Theory of Attributed Equivalence in Databases with Application to Schema Integration. *IEEE Trans. Softw. Eng.*, 15(4):449–463, 1989.

[97] Claudia Leacock, George A. Miller, and Martin Chodorow. Using Corpus Statistics and WordNet Relations for Sense Identification. *Comput. Linguist.*, 24(1):147–165, 1998.

[98] Yoonkyong Lee, Mayssam Sayyadian, AnHai Doan, and Arnon S. Rosenthal. Tuning Schema Matching Software using Synthetic Scenarios. *The VLDB Journal*, 16(1):97–122, 2007.

[99] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Transactions on Knowledge and Data Engineering*, 21(8):1218–1232, 2009.

[100] Wen-Syan Li and Chris Clifton. Semantic Integration in Heterogeneous Databases Using Neural Networks. In *VLDB Proceedings*, pages 1–12, 1994.

[101] L. Lovász and M.D. Plummer. *Matching Theory*. Akadémiai Kiadó, Budapest, 1986.

[102] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Halevy. Corpus-Based Schema Matching. In *ICDE Proceedings*, pages 57–68, 2005.

[103] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic Schema Matching with Cupid. In *VLDB Proceedings*, 2001.

[104] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA - A MApping FRAmework for Distributed Ontologies. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 235–250, 2002.

[105] Ming Mao, Yefei Peng, and Michael Spring. A Profile Propagation and Information Retrieval Based Ontology Mapping Approach. In *Proceedings of the Third International Conference on Semantics, Knowledge and Grid*, pages 164–169, 2007.

[106] 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*, pages 336–342, 2008.

[107] Anan Marie and Avigdor Gal. Managing uncertainty in schema matcher ensembles. In *Proceedings of the 1st international conference on Scalable Uncertainty Management*, pages 60–73, 2007.

[108] Anan Marie and Avigdor Gal. On the Stable Marriage of MaximumWeight Royal Couples. In *Proceedings of AAAI Workshop on Information Integration on the Web (IIWeb'07)*, 2007.

[109] Anan Marie and Avigdor Gal. Boosting schema matchers. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE*, pages 283–300, 2008.

[110] Sabine Massmann and Erhard Rahm. Evaluating Instance-based Matching of Web Directories. In *WebDB Proceedings*, 2008.

[111] Paul McNamee. Language identification: a solved problem suitable for undergraduate instruction. *J. Comput. Sci. Coll.*, 20(3):94–101, 2005.

[112] Giansalvatore Mecca, Paolo Papotti, and Salvatore Raunich. Core Schema Mappings. In *SIGMOD Proceedings*, pages 655–668, 2009.

[113] C. Meilicke, H. Stuckenschmidt, and Andrei Tamilin. Repairing Ontology Mappings. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, pages 1408–1413, 2007.

[114] Christian Meilicke and Heiner Stuckenschmidt. Analyzing Mapping Extraction Approaches. In *Proceedings of the 2nd International Workshop on Ontology Matching (OM-2007)*, 2007.

[115] Christian Meilicke and Heiner Stuckenschmidt. Applying Logical Constraints to Ontology Matching. In *Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*, pages 99–113, 2007.

[116] Christian et al. Meilicke. A Reasoning-Based Support Tool for Ontology Mapping Evaluation. In *ESWC Proceedings*, pages 878–882, 2009.

[117] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *ICDE Proceedings*, pages 117–128, 2002.

[118] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm (Extended Technical Report). Technical Report 2001-25, Stanford InfoLab, June 2001.

[119] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[120] Malgorzata Mochol and Anja Jentzsch. Towards a Rule-Based Matcher Selection. In *Proceedings of the 16th international conference on Knowledge Engineering: Practice and Patterns*, pages 109–119, 2008.

[121] Malgorzata Mochol, Anja Jentzsch, and Jérôme Euzenat. Applying an Analytic Method for Matching Approach Selection. In *In Proceedings of the The First International Workshop on Ontology Matching*, 2006.

[122] Alvaro Monge and Charles Elkan. The Field Matching Problem: Algorithms and Applications. In *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[123] A.H. Nezhadi, B. Shadgar, and A Osareh. Ontology Alignment Using Machine Learning Techniques. *International Journal of Computer Science & Information Technology*, 3(2):139, 2011.

[124] Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In *IJCAI Proceedings*, 2011.

[125] Henrik Nottelmann and Umberto Straccia. Information retrieval and machine learning for probabilistic schema matching. *Inf. Process. Manage.*, 43(3):552–576, 2007.

[126] Natalya F. Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 450–455, 2000.

[127] Natalya F. Noy and Mark A. Musen. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 63–70, 2001.

[128] Natalya F. Noy and Mark A. Musen. The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. *Int. J. Hum.-Comput. Stud.*, 59:983–1024, 2003.

[129] Luigi Palopoli, Giorgio Terracina, and Domenico Ursino. Dike: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Softw. Pract. Exper.*, 33:847–884, 2003.

[130] Jyotishman Pathak and Christopher G. Chute. Debugging Mappings between Biomedical Ontologies: Preliminary Results from the NCBO BioPortal Mapping Repository. In *Proceedings of the International Conference on Biomedical Ontology*, 2009.

[131] Heiko Paulheim. On Applying Matching Tools to Large-scale Ontologies. In *Proceedings of the 3rd International Workshop on Ontology Matching (OM-2008)*, volume 431, 2008.

[132] S. Pavel and J. Euzenat. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on Knowledge and Data Engineering*, PP(99):158 – 176, 2011.

[133] Eric Peukert. Modeling matching systems using matching process design patterns. In *OM Proceedings*, 2011.

[134] Eric Peukert, Henrike Berthold, and Erhard Rahm. Rewrite Techniques for Performance Optimization of Schema Matching Processes. In *EDBT Proceedings*, pages 453–464, 2010.

[135] Eric Peukert, Julian Eberius, and Erhard Rahm. AMC - A Framework for Modelling and Comparing Matching Systems as Matching Processes. In *ICDE Proceedings*, pages 1304–1307, 2011.

[136] Eric Peukert, Julian Eberius, and Erhard Rahm. A Self-Configuring Schema Matching System. In *ICDE Proceedings*, pages 306–317, 2012.

[137] Eric Peukert, Sabine Massmann, and Kathleen König. Comparing Similarity Combination Methods for Schema Matching. In *GI Jahrestagung (1)*, pages 692–701, 2010.

[138] Eric Peukert and Erhard Rahm. Restricting the Overlap of Top-N Sets in Schema Matching. In *Proceedings of the 1st Workshop on New Trends in Similarity Search*, pages 20–25, 2011.

[139] Giuseppe Pirra and Domenico Talia. UFOme: An ontology mapping system with strategy prediction capabilities. *Data and Knowledge Engineering*, 69(5):444–471, 2010.

[140] Erhard Rahm. *Schema Matching and Mapping*, chapter Towards large-scale schema and ontology matching. Springer-Verlag, 2011.

[141] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10:334–350, 2001.

[142] Erhard Rahm, Hong-Hai Do, and Sabine Massmann. Matching Large XML Schemas. *SIGMOD Record*, 33:26–31, 2004.

[143] Salvatore Raunich and Erhard Rahm. ATOM: Automatic Target-driven Ontology Merging. In *ICDE Proceedings*, pages 1276–1279, 2011.

[144] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979.

[145] George G. Robertson, Mary P. Czerwinski, and John E. Churchill. Visualization of Mappings Between Schemas. *SIGCHI*, pages 431–439, 2005.

[146] Stephen E. Robertson and Karen Sparck Jones. Document retrieval systems. chapter Relevance weighting of search terms, pages 143–160. Taylor Graham Publishing, 1988.

[147] Cornelius Rosse and José L. V. Mejino Jr. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *Journal of Biomedical Informatics*, 36(6):478–500, 2003.

[148] Seung Hwan Ryu, Boualem Benatallah, Hye-Young Paik, Yang Sok Kim, and Paul Compton. Similarity Function Recommender Service Using Incremental User Knowledge Acquisition. In *Proceedings of the 9th international conference on Service-Oriented Computing*, pages 219–234, 2011.

[149] Barna Saha, Ioana Stanoi, and Kenneth L. Clarkson. Schema Covering: a Step Towards Enabling Reuse in Information Integration. In *ICDE Proceedings*, pages 285–296, 2010.

[150] Khalid Saleem, Zohra Bellahsene, and Ela Hunt. PORSCHE: Performance ORiented SCHEma mediation. *Inf. Syst.*, 33:637–657, 2008.

[151] SAP. Warp10 community-based integration. (white paper), 2010. https://cw.sdn.sap.com/cw/docs/DOC-120470 [last visited January 2013].

[152] Len Seligman, Peter Mork, Alon Y. Halevy, Kenneth P. Smith, Michael J. Carey, Kuang Chen, Chris Wolf, Jayant Madhavan, Akshay Kannan, and Doug Burdick. OpenII: An Open Source Information Integration Toolkit. In *SIGMOD Proceedings*, pages 1057–1060, 2010.

[153] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[154] Feng Shi, Juanzi Li, Jie Tang, Guotong Xie, and Hanyu Li. Actively Learning Ontology Matching via User Interaction. In *ISWC Proceedings*, pages 585–600, 2009.

[155] Pavel Shvaiko and Jérôme Euzenat. A Survey of Schema-Based Matching Approaches. *Journal on Data Semantics IV*, 3730:146–171, 2005.

[156] Marko Smiljanic. *XML schema matching : balancing efficiency and effectiveness by means of clustering*. PhD thesis, Enschede, 2006.

[157] Marko Smiljanic, Maurice van Keulen, and Willem Jonker. Using Element Clustering to Increase the Efficiency of XML Schema Matching. In *ICDE Workshops Proceedings*, 2006.

[158] T. Smith and M. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[159] Weifeng Su, Jiying Wang, and Frederick Lochovsky. Holistic Schema Matching for Web Query Interfaces. In *EDBT Proceedings*, pages 77–94, 2006.

[160] Gabriele Taentzer and Martin Beyer. Amalgamated Graph Transformations and Their Use for Specifying AGG - an Algebraic Graph Grammar System. In *In Proceedings of the Dagstuhl Seminar on Graph Transformations in Computer Science*, pages 380–394, 1993.

[161] He Tan and Patrick Lambrix. A Method for Recommending Ontology Alignment Strategies. In *ISWC Proceedings*, pages 494–507, 2007.

[162] Samir Tartir, I. Budak Arpinar, Michael Moore, Amit P. Sheth, and Boanerges Aleman-meza. OntoQA: Metric-based ontology quality analysis. In *In Proceedings of the IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005.

[163] Axel Tenschert, Matthias Assel, Alexey Cheptsov, Georgina Gallizo, Emanuele Della Valle, and Irene Celino. Parallelization and Distribution Techniques for Ontology Matching in Urban Computing Environments. In *In Proceedings of the 4th International Workshop on Ontology Matching (OM-2009)*, volume 551, 2009.

[164] Bach Thanh Le and Rose Dieng-Kuntz. A Graph-Based Algorithm for Alignment of OWL Ontologies. In *In Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 466–469, 2007.

[165] Andreas Thor. Toward an adaptive String Similarity Measure for Matching Product Offers. In *GI Jahrestagung (1)*, pages 702–710, 2010.

[166] Andreas Thor, Toralf Kirsten, and Erhard Rahm. Instance-based matching of hierarchical ontologies. In *Proceedings of 12. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW)*, pages 436–448, 2007.

[167] Andreas Thor and Erhard Rahm. MOMA - A Mapping-based Object Matching System. In *Proceedings 3rd Conference on Innovative Data Systems Research (CIDR)*, 2007.

[168] KeWei Tu and Yong Yu. CMC: Combining Multiple Schema-Matching Strategies Based on Credibility Prediction. In *Proceedings of the 10th international conference on Database Systems for Advanced Applications*, pages 888–893, 2005.

[169] Konrad Voigt. *Structural graph-based metamodel matching*. PhD thesis, Dresden University of Technology, 2011.

[170] Konrad Voigt and Thomas Heinze. Metamodel Matching Based on Planar Graph Edit Distance. In *Proceedings of the Third international conference on Theory and practice of model transformations*, pages 245–259, 2010.

[171] Konrad Voigt, Petko Ivanov, and Andreas Rummler. Matchbox: combined metamodel matching for semi-automatic mapping generation. In *ACM Symposium on Applied Computing (SAC)*, pages 2281–2288, 2010.

[172] W3C. Semantic Web Services Ontology (SWSO), 2005. http://www.w3.org/Submission/SWSF-SWSO/ [last visited may 19th 2013].

[173] Weka Machine Learning Project. Weka. http://www.cs.waikato.ac.nz/˜ml/weka [last visited may 19th 2013].

[174] William E. Winkler. The State of Record Linkage and Current Research Problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.

[175] Zhibiao Wu and Martha Palmer. Verb Semantics And Lexical Selection. In *In Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138, 1994.

[176] Wenwei Xue, Hungkeng Pung, Paulito P. Palmes, and Tao Gu. Schema Matching for Context-Aware Computing. In *In Proceedings of the 10th international conference on Ubiquitous computing*, pages 292–301, 2008.

[177] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Syst. Man Cybern.*, 18(1):183–190, 1988.

[178] K. Yang and R. Steele. A Framework for Ontology Mapping for the Semantic Web. In *In Proceedings of the International Conference on Information Technology in Asia*, 2008.

[179] Qing Yang, Li Zhu, and Wei Chen. Research on Ontology Matching Method Based on Description Logics Reasoning Mechanism. In *In Proceedings of the 2009 International Conference on Web Information Systems and Mining*, pages 209–212, 2009.

[180] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.

[181] K. Zhang and D. Shasha. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[182] Anna V. Zhdanova and Pavel Shvaiko. Community-Driven Ontology Matching. In *ESWC Proceedings*, pages 34–49, 2006.

[183] Z. Zhen, J. Shen, J. Zhao, and J. Qian. LiSTOMS: a Light-weighted Self-tuning Ontology Mapping System. In *IEEE/WIC/ACM Proceedings*, 2010.

[184] Hai Zhuge. A process matching approach for flexible workflow process reuse. *Information & Software Technology*, 44(8):445–450, 2002.