Development and Evaluation of a Flexible Methodology for Ontology Matching

Katrin Eisenreich
Born on December 29th, 1983 in Rodewisch

Supervised by:
Dr.-Ing. Thomas Springer (TU Dresden)
Dr. Gregor Hackenbroich (SAP AG)

Submitted on August 31, 2007
Aufgabenstellung

Text der Aufgabenstellung...
Confirmation

I hereby confirm that I have entirely independently prepared this thesis with the topic

Development and Evaluation of a Flexible Methodology for Ontology Matching

and that I used only the references and auxiliary means listed in the bibliography and properly cited all literature references.

Dresden, August 31, 2007

Katrin Eisenreich
TO TONATIUh AND MY PARENTS.
Abstract

Today, ontologies play a crucial role in a variety of IT scenarios. Above all, they are applied increasingly in highly distributed and heterogeneous environments, where they are considered a key factor for interoperability and knowledge sharing. However, since different parties in an open environment tend to commit to different ontologies, the problem of heterogeneity arises anew — on the level of ontologies. Ontology matching tries to bridge this heterogeneity by determining correspondences between elements from two ontologies.

In this thesis, we address the development of a semi-automatic methodology for ontology matching. To this end, we first investigate the current state of the art of ontology matching approaches. Based on the insights from this analysis, we devise a new methodology incorporating different aspects we regard as promising approaches for effective ontology matching. We argue that in order to produce high-quality matching results, we must leverage the broad spectrum of semantic information available in ontologies. Furthermore, we observe that a highly configurable matching process is decisive for allowing users to tailor the methodology to specific matching tasks. Based on this premise, we devise a framework-based solution to the ontology matching problem, which enables users to apply and integrate a range of approaches in a modular way. Furthermore, we present functionality for automatic matching configuration to account for application scenarios where no manual configuration is possible.

We implement the presented methodology in the prototype COMA++(O), which is based on the schema matching tool COMA++ [Do05]. We extend the existing tool with an internal representation of ontologies accounting for their expressive semantic features and realize a set of ontology-specific matchers based on this architectural grounding. We test the prototype against a series of ontology matching tasks published by the Ontology Alignment Evaluation Initiative (OAEI) [OAE07], which constitutes the most renowned benchmark for ontology evaluation hitherto existing.

The results of the conducted evaluation indicate both the effectiveness and generic applicability of our methodology. In particular, we achieve very high matching qualities ranking COMA++(O) among the best state-of-the-art systems for ontology matching, such as RiMOM [YLT06] and Falcon-AO [JHCQ05]. We hold that the methodology can achieve those high-quality results both in generic and customized matching settings due to its high flexibility.
Contents

1 Introduction 5
  1.1 Motivation and Objective .............................................. 5
  1.2 Application Scenarios ................................................... 7
  1.3 Ontologies and Ontology Matching in Research and Industry .......... 8
    1.3.1 The THESEUS Research Programme ................................ 8
    1.3.2 SUPER — Semantics Utilized for Process Management ............ 9
    1.3.3 Visibility and Prospects of Ontologies in Research and Industry 10
  1.4 Terms and Definitions .................................................. 10
  1.5 Outline ................................................................. 12

2 State of the Art in Ontology Matching 13
  2.1 Schema Matching vs. Ontology Matching ............................ 13
  2.2 Ontology Matching in Different Integration Scenarios ............... 15
  2.3 Categorization of Ontology Mismatches ................................ 17
  2.4 Ontology Languages and Language-Level Mismatches .................. 18
    2.4.1 OWL — A DL-based Ontology Language ............................ 19
    2.4.2 F-Logic — A Frame-based Ontology Language .................... 20
    2.4.3 Mismatches between OWL and F-Logic ............................. 20
    2.4.4 Conclusion .......................................................... 21
  2.5 The Matching Process ................................................... 22
    2.5.1 A Generic Matching Process ....................................... 22
    2.5.2 Matching Input: Information Utilized in the Matching Process ... 22
    2.5.3 Matching Output: Representation of Discovered Mappings ........ 24
  2.6 Approaches Applied for Discovery of Mappings ....................... 25
    2.6.1 Ontology-based Approaches ....................................... 26
      2.6.1.1 Element-Level Approaches ................................ 27
      2.6.1.2 Structure-Level Approaches ................................ 28
    2.6.2 Instance-based approaches ........................................ 30
      2.6.2.1 Element-Level Approaches ................................ 30
      2.6.2.2 Probabilistic Approaches ................................... 30
    2.6.3 Community-driven Ontology Matching ................................ 31
  2.7 Representative Tools .................................................... 31
    2.7.1 OLA ................................................................. 31
    2.7.2 RiMOM ............................................................... 32
    2.7.3 Falcon-AO ........................................................... 32
    2.7.4 An Information Retrieval Approach — iMapper .................... 32
  2.8 Summary ................................................................. 33

3 Towards a Flexible Ontology Matching Methodology 35
  3.1 Design Goals ............................................................ 35
  3.2 Concept Outline ........................................................ 37
  3.3 Definition of Element Descriptions ................................... 39
    3.3.1 Description Features of Ontology Elements ....................... 40
4 Implementation of the Methodology
4.1 COMA++ — A Schema Matching Framework
4.1.1 Architecture and Functionality Overview
4.1.2 The Matching Process in COMA++
4.1.3 COMA++ as Basis for Flexible Ontology Matching
4.2 Integrating the Ontology Graph Representation
4.2.1 Implementation of the Ontology Graph Representation
4.2.2 Parsing, Import and Loading of Ontologies
4.3 Matching Strategies for Integration of Individual Matchers
4.3.1 DependentMatcher
4.3.2 StrategyMatcher
4.3.3 PropagationMatcher
4.4 Extending the Matcher Library
4.4.1 Basic Matchers
4.4.2 The Lexical Matcher
4.4.3 Composite Matchers
4.4.4 Applying the Matching Strategies
4.5 Graphical User Interface
4.5.1 Visualization of Ontologies
4.5.2 Configuration Facilities
4.6 Summary

5 Evaluation of the Prototype
5.1 Evaluation Procedure and Measures
5.2 Evaluation Test Base
5.2.1 Test Ontologies and Task Series
5.2.2 Contest-based Evaluation
5.3 Evaluation Objectives and Experiment Design
5.4 Experiment Results
5.4.1 Customized Matching Evaluation
5.4.2 Generic Matching Evaluation
5.5 Analysis and Assessment of Individual Approaches
5.5.1 Combining Constituent Matchers
1 Introduction

During the last decade, ontologies and applications that exploit ontological knowledge have been attracting increasing attention from various fields such as information integration, knowledge management, information retrieval, and e-commerce. Today, concepts such as the Semantic Web and Service Oriented Architecture (SOA) above all highlight the need for interoperability within a highly heterogeneous distributed environment as is the web. The semantics of information sources, web content, or web services can be described formally by ontologies, providing a common agreed-on vocabulary that can be used for compliant requests.

For complex requests to such semantic information, it is necessary to combine knowledge described across several ontologies. In such cases, heterogeneity between ontologies often impedes direct inference over the knowledge they represent. This raises the need to reconcile, or integrate, the ontologies. Since most real-world scenarios depend on heterogeneous ontologies, the issue of ontology reconciliation indeed plays a crucial role to tap the full potential of ontology-based applications.

A central problem in all ontology reconciliation tasks is that of ontology matching. The goal of ontology matching is the discovery of semantic correspondences, so-called ontology mappings, between elements of two ontologies. Such correspondences can then be leveraged in ontology translation or integration tasks. In this work, we develop a methodology for (semi-)automatic ontology matching. We elaborate a framework-based solution of the underlying process and present an implementation of our methodology based on an existing infrastructure. A comprehensive evaluation is carried out to analyze the quality of the resulting tool and to compare it to other state-of-the-art approaches.

1.1 Motivation and Objective

Initially, ontologies have been proposed as a means of knowledge representation and sharing in AI. In the last decade, the focus has shifted towards much broader, distributed, and more open application environments, such as in the fields of information integration, knowledge management systems, e-commerce, and the Semantic Web. For a long time, researchers have regarded ontologies as the solution to interoperability problems that arise in such environments, since they provide a common vocabulary through which agents of a domain can communicate. However, it has now become a generally acknowledged fact in research and industry that for most application domains we will not achieve a complete commitment to any one standard ontology. Reasons for that can be pragmatic, that is, rooted in different application- or task-specific requirements; cultural, that is, based on different conceptions of certain aspects of the world; or simply because of commitment to well-established ontologies, for example, within a company. Thus, rather than providing a common agreed-on vocabulary, ontologies have evolved as a new source of heterogeneity in system landscapes.

Improving the performance of knowledge sharing and knowledge-based system interoperability therefore depends on a large extent on functionality for reconciliation and integration between heterogeneous ontologies. A central task in the process of reconciliation consists in the discovery of correspondences between elements of two ontologies. We refer to this task as ontology matching.

A simple example to illustrate the ontology matching problem is given in Figure 1.1. The two ontologies shown here could stem from an e-commerce scenario, where they represent the product catalogues of two different online stores, each describing the products of the respective store as concepts, and the relations between the products as properties of those concepts. Looking at the two ontologies, we can
easily identify mappings (correspondences) between concepts and properties in the left ontology and those in the right one. In the figure, we indicate possible mappings by dashed lines between the ontology elements. Provided with such a set of mappings, we can now perform translation between the ontologies, or integrate them into one. This way, we could, for example, provide agents (i.e., users, applications) with an integrated view on both product catalogues, thus enabling them to submit queries to an integrated marketplace.

Figure 1.1: Example matching task

To this date, matching of ontologies is often still largely performed manually, which is a tedious and time-consuming task. Naturally, we want to minimize the manual effort that must be invested into the discovery of mappings. Ontology matching systems try to do so by automating (part of) the matching process.

Ontology matching practice today encompasses a variety of fields, such as database schemata, heuristics, linguistics, and machine learning approaches. Indeed, the problem as such is closely related to that of schema matching, which has long been in the focus of research in diverse database application domains such as data integration and warehousing, query processing, and e-commerce. A great variety of approaches for the matching of database schemata have been developed. A comprehensive overview and classification of such approaches can be found in [RB01, SE05]. Many techniques applied in the field of schema matching can be utilized for ontology matching in an analogue manner. Still, there are important differences that distinguish the problem of ontology matching from its counterpart in the database domain, thereby creating the need for dedicated ontology matching approaches. Those aspects will be discussed in Section 2.1.

The aim of this work is to develop a methodology for (semi-)automatic ontology matching that flexibly leverages the various features contained in ontologic descriptions. To this end, we evaluate the state of the art in ontology matching to get an understanding of the different techniques currently applied, and to identify possible benefits and shortcomings of individual approaches. On this basis, we will choose a number of approaches we deem valuable contributions to our work, incorporating them in our ontology matching methodology. Ultimately, we aim to implement the devised methodology in a prototype that performs well compared to current state-of-the-art tools for ontology matching.

A tool developed in the database domain to facilitate semi-automatic matching of schemata is COMA++ [Do05]. The tool enables the combination of different schema matching approaches based on a generic framework architecture. We argue that a flexibly configurable matching process can lead to good quality matching results, since it allows a user to account for the widely diverse characteristics of ontology matching tasks. Based on this hypothesis, we develop a flexible methodology for ontology matching and implement it as an extension to the framework architecture of COMA++.

In order to provide for a well-founded assessment of the matching quality provided by our methodology,
we carry out a comprehensive evaluation. This evaluation shall be based on a series of ontology matching tasks published as a benchmark for ontology matching algorithms. Based on the evaluation results, we want to compare the performance of our solution with other ontology matching systems. This way, we can assess merits and downsides of the proposed methodology and thus gain insights as to how we can improve the ontology matching process in future.

### 1.2 Application Scenarios

To further demonstrate the need for effective ontology matching and its high potential in various areas, this section describes a number of possible application scenarios.

- **Catalog integration for B2B marketplaces**
  
  One of the most widely known tasks where ontology matching can play an important role is the problem known as *catalogue integration*, which we have illustrated in the example scenario in Section 1.1. The product range of a company can be described in terms of ontologies, that is, by specifying products and possible relations between them as ontology concepts and properties, thus making explicit the semantics of products. The ontologies can then be used in Business-To-Business (B2B) applications to automatically retrieve information about a product based on queries that go beyond traditional keyword-based approaches. Obviously, it can not be assumed that all retailers or producers commit to a single shared ontology, considering the fast change in technology, let alone the diversity in different industry sectors, or cultural backgrounds. Therefore, to be able to use such ontologies in a combined way, we first have to make them compatible. Having aligned the catalogue ontologies, we could provide users with an integrated marketplace. Then, they could retrieve and compare product information from several stores by running a single query using ontology concepts from either of the ontologies.

- **Agent interaction**
  
  Agents that interact in a multi-agent system must communicate in order to fulfil their goals, for example, to trade goods in the B2B marketplace scenario presented above. Agent communication takes place by exchanging messages in a given language, which can be described using ontologies. In most cases, such ontological descriptions will be proprietary and specific to a particular agent. Concepts from two agents’ ontologies are therefore likely to be heterogeneous and must be related first in order to enable the agents to “understand” each other. Moreover, with open systems where agents enter and leave dynamically, the problem of ontology heterogeneity and the need for an effective matching approach get even more evident.

- **Collaborative knowledge acquisition on the Semantic Web**
  
  Ontologies can be used to convey meaning — semantics — to information published on the web. This idea has entered the focus of attention in recent years by the term *Semantic Web*. The development of such information and the corresponding semantic specifications (ontologies) will take place in a highly decentralized and distributed environment. Thus, the resulting ontologies will inevitably be semantically disparate, even when describing the same domain in a similar application context. To enable knowledge acquisition across multiple information sources (e.g., semantically annotated web pages), the ontologies assigned to those sources need to be reconciled first in order to allow reasoning on their semantics.

- **Enabling complex business processes through automated web service orchestration**
  
  Web services are applications that are uniquely identified by an URI and can be invoked using standard internet protocols, their interfaces being described in XML. Rather than viewing those services as isolated components, the idea is to compose them into complex processes depending on a customer’s needs and preferences.
  
  In an effort to further automate discovery, selection, and combination of services, researchers aim
to describe the semantics of web services [SWS07, BDMW07] based on dedicated ontologies. With the help of such Semantic Web Services (SWS), an enterprise that requires to interact with another enterprise could automatically select an appropriate web service based on the provided data and the underlying web service and process ontologies.

As in the scenarios above, there is a high potential for heterogeneity between such web service ontologies, and their reconciliation is a prerequisite for an integrated view on the web service semantics. We will refer back to this scenario below, describing it in the context of current research endeavours.

- **Collaborative product development**
  
  For higher competitiveness, shorter product development cycles, and better response to market demands, large companies need to manage information about their products and production processes throughout their entire lifetime. To achieve this task, Product Lifecycle Management (PLM) systems need a holistic view on application systems, product related data, processes, and people. Researchers who investigated this problem (see, e.g., [MBB05, BMM+06]) have suggested to integrate different information sources semantically via a shared ontology. An ontology-based content management system applying this idea is presented in [MISA03]. The authors point out how the semantically enriched content management increases the facility of maintaining and accessing content, as well as deriving value from potential synergies between different contents.

  In such a system, extensions (i.e., specializations) of the shared ontology could be applied to facilitate integration of specific semantics occurring, for example, in different departments or within different phases of the lifecycle. Heterogeneous extensions evolving within companies, or even the need for interoperability across company boundaries (i.e., between two company-specific PLM ontologies) would require a suitable matching mechanism.

While the discussed scenarios agree in the need for reconciliation between participating ontologies, the specific requirements and prerequisites of reconciliation can differ considerably depending on the application context. For example, we might have to accept some trade-off between the efficiency of the applied algorithms and the quality of the obtained alignment. This becomes evident when considering, for example, the different requirements of static environments such as in the catalogue integration scenario, and dynamic environments such as in the scenario of agent communication. We will resume this issue in Section 2.2.

### 1.3 Ontologies and Ontology Matching in Research and Industry

As illustrated above, ontologies and the problem of reconciling them are relevant today in many areas. A comprehensive evaluation of possible application scenarios is out of the scope of this thesis. Rather, we focus our attention on two current research projects in the field of SOA and BPM, in which SAP is a major participant. We argue that due to the inherent heterogeneity of the underlying environments, the application of high-quality semi-automatic matching approaches could considerably benefit the presented projects.

#### 1.3.1 The THESEUS Research Programme

THESEUS [The07] is a large-scale research program initiated and subsidized by the German Federal Ministry of Economics and Technology. The focus of THESEUS is on the development of semantic technologies and their integration in a new internet-based knowledge infrastructure. In such an environment, machines shall be enabled to recognize and classify the semantics (meaning) of data, and to infer new knowledge based on existing information from various sources. In that context, ontologies and their management will play a central role.
1.3. ONTOLOGIES AND ONTOLOGY MATCHING IN RESEARCH AND INDUSTRY

Under the umbrella of THESEUS, industrial and public research partners are collaborating to develop new application-oriented technologies and technical standards and to test them in different application scenarios. At the current time, the involved partners include a total of 30 institutions and companies, such as the Dresden University of Technology, the Fraunhofer-Gesellschaft, SAP, and Siemens. One of the practical scenarios that shall be used as a test bed for new technologies is the SAP-led application scenario TEXO.

TEXO — The Services Internet  Based on the concept of SOA, TEXO aims to provide businesses with a novel infrastructure for web-based applications, enabling the creation, provision, distribution, selection, coordination, and composition of services. Extending traditional SOA solutions, the TEXO use case focuses on the realization of business value networks. In those networks, business shall be enabled to interact efficiently with their customers and suppliers by leveraging available services flexibly. In such a highly heterogeneous and dynamic “internet of services”, services must be described semantically in order to allow consumers to understand their application context, intended purpose, and use. Depending on the application, the required information comprises technical, commercial, social, and legal aspects. Ontologies and other semantic tools for description of web services and other components are therefore at the core of the described service provider infrastructure.

The project goals of TEXO comprise the development of a dedicated business ontology framework, together with corresponding tools for ontology management in service-oriented environments. Facing the potential diversity of participants in the internet of services, integration with heterogeneous ontologies (e.g., various company-specific ontologies) will be a strong prerequisite for wide acceptance of the developed standards. Therefore, matching between ontologies will be one key factor to tap the potential of the semantic techniques applied.

1.3.2 SUPER — Semantics Utilized for Process Management

The SUPER project [Sup07], a member of the European Semantic Systems Initiative (ESSI) cluster[^1] and financed from the European Union 6th Framework Programme, aims at raising the task of Business Process Management (BPM) from the IT to the business level [BDMW07]. That is, control of business processes shall be shifted away from the rather involved technical level where it is mostly conducted nowadays, to the level of business decisions and management. SUPER wants to achieve this by providing a framework for Semantic Business Process Management (SBPM) [HLD05].

SBPM is a novel approach that tries to increase the level of automation in BPM by enabling automated translation between enterprise process spaces. To this end, SBPM applies semantic (ontological) representations of both the business perspective, that is, the business requirements a process shall address, and the systems perspective (e.g., the involved resources and roles). Based on the semantic representations, standard reasoning techniques can be applied to translate between the two perspectives.

Following this idea, the SUPER framework is based on traditional BPM in combination with SWS [BDMW07], that is, web services that come with a well defined specification of their semantics, based on some ontological description. Ontologies are used throughout the different layers of the framework to describe the core business aspects of SBPM. The applied ontologies comprise both pre-existing ontologies such as the Web Service Modelling Ontology (WSMO) [ESS07], which is used to describe aspects of Web Services, and ontologies developed specifically within SUPER [BCD+07], such as the Business Process Modelling Ontology (BPMO), or the Process Mining Ontology (PMO). Furthermore, the framework includes a set of organizational-related ontologies which model concepts like Organization, Team, Employee, or Role and the relationships between them. For a detailed description of the Business Process Ontology Framework, see [BCD+07].

[^1]: http://www.essi-cluster.org/
Ontology matching in SBPM  The described ontology framework provides a basis for representing the semantics of web services within SUPER. However, a system that shall be integrated with this framework might use different ontologies to describe its resources (i.e., its information assets, organizational structure, etc.). This raises the problem of semantic heterogeneity between participants in the SUPER framework. Hence, we need to provide functionality for discovery of and translation between corresponding ontology elements to achieve interoperability between the ontologies. Within SUPER, this functionality is provided by means of so-called Mediators [BCD+07]. Mediators define the mappings that exist between heterogeneous ontologies and the translation rules required to transform an element to its corresponding element in another ontology. At present, only rudimentary ontology matching approaches, which exploit the similarity of concept labels, are applied in the framework. A more advanced (semi-)automatic ontology matching mechanism could benefit the SUPER framework by supporting high-quality mapping discovery, thus facilitating more efficient Mediator construction.

1.3.3 Visibility and Prospects of Ontologies in Research and Industry

The topic of ontology matching has established itself as an independent field of research, featuring a stable community, whose members are rooted in various related areas. Consequently, matching methodologies have been developed with focus on many diverse application contexts, taking into account different requirements that arise in such contexts. Moreover, researchers have worked towards a holistic view on and a precise definition of the ontology matching problem, and have investigated into objective evaluation methods for ontology matching systems. In this context, the Ontology Alignment Evaluation Initiative (OAEI) [OAE07] has initiated a yearly evaluation campaign, where ontology matching systems can be evaluated against a set of ontology matching tasks. These include systematically generated tasks from the domain of bibliographic reference and varying tasks that require matching of large-sized real-world ontologies. At the time being, those are largely ontologies that model some taxonomy. The focus of the campaign is mainly on the competition between different matching approaches in order to investigate their strengths and weaknesses. It is not intended as a resource of different domain-specific matching tasks. In particular, the campaign does not feature any ontology from the business domain. As the previously described projects show, the prospect that lies in the application of ontologies becomes now acknowledged beyond an isolated research community, and starts to enter the industrial sector. In this context, ontologies are principally considered as an enabling factor for flexible, knowledge-based business integration systems. However, while ontologies in other fields such as the domain of biology and medicine (e.g., [GO207]) have been developed and applied for many years and are available to the public, this is still not the case for ontologies in the business domain. It is yet an open issue whether this will change during the next years, the more so as companies might be less inclined to publish their knowledge assets. The answer to that question will to a large degree set the course for the development of ontology management tools in the business domain, including techniques for ontology matching.

1.4 Terms and Definitions

In the literature, the terminology referring to what we call ontology matching and to different adjacent tasks is often quite blurred. For this reason, we shall first provide a short overview of different terms and define their intended meaning throughout this document.

Ontology  The arguably most commonly used definition of an ontology in the field of computing introduces the term as an “explicit specification of a conceptualization” [Gru93] of some domain. In other words, an ontology models the knowledge we have about a specific part of the world by means of formal constructs such as concepts that relate to certain aspects of the modelled domain, and properties that represent relations between the concepts. Ontologies make explicit the semantics of terms (i.e.,
syntactic descriptions) that refer to things within the domain. Systems can use one or several (combined) ontologies to reason over the specified semantics of the underlying domain(s) of interest.

**Ontology Reconciliation** Ontologies have initially been conceived as the silver bullet for enabling system interoperability. However, in reality it turns out that mostly there exists some kind of heterogeneity between ontologies that shall be applied in a given scenario. We use the terms coordination or reconciliation of ontologies when talking about any problem where knowledge represented by two ontologies needs to be reconciled in order to exploit their semantics in a combined way.

**Ontology Mapping** When aiming at a reconciliation between two ontologies, we first need to discover pairs of elements from the two ontologies that correspond to each other, that is, such elements that are semantically equal (or similar). Such a correspondence is called a mapping. A mapping is in general associated with a mapping expression and a confidence value. The mapping expression is a function stating the precise relation that is assumed between the two mapped elements, such as equivalence or subsumption between classes or properties. Indeed, it can be an arbitrary logical expression, either derived automatically or provided by a user. The confidence value is some value in the interval \([0 \ldots 1]\), which indicates the degree of belief that the mapped elements indeed correspond to each other.

**Ontology Matching and Alignment** The terms ontology matching and alignment are often used interchangeably. For the sake of clarity, we will use the former when referring to the process of mapping discovery as a whole, while the latter is used to denote the result of this process, that is, the complete set of discovered mappings.

The matching process can be described as follows. Given two ontologies \(o\) and \(o'\) (input), find an alignment \(A\) (output) between the two ontologies, that is, find the mappings between elements in the two ontologies that correspond to each other semantically. We observe that there are many other aspects that can influence the matching process. Therefore, following the definition given in [BEE+04], we define the matching process as illustrated in Figure 1.2:

**The matching process** is a function \(f\) which, given a pair of ontologies \(o\) and \(o'\), a (possibly empty) input alignment \(A_{in}\), a set of parameters \(p\), and a set of auxiliary resources \(r\), returns an alignment \(A\) between the input ontologies. \(A_{in}\) could, for instance, be a set of mappings provided in advance by the user or as the result of a previous matching process. An auxiliary resource could, for example, be a list of synonym words. Parameters \(p\) influence how the process is conducted, possibly allowing a user to customize the matching.

**Ontology Merging, Translation and Integration** The process of ontology matching forms the first step in several other, more advanced tasks in the field of ontology reconciliation.
One possible application of an alignment, which is obtained as result of a matching process, is the translation between ontologies, that is, the translation of sentences that use concepts from a source ontology into sentences that use concepts of a target ontology. Translation is conducted by using the expressions associated with the discovered mappings. Another task is that of merging, which is the process of building a single ontology from two given source ontologies, while preserving the semantics as defined in the source ontologies. For translation, as well as for merging of ontologies, the domains over which the ontologies are defined must be similar or overlapping to a reasonably large degree. Yet another term is that of ontology integration, which refers to the composition of two ontologies into one, with the respective vocabularies usually not being defined over the same domain.

1.5 Outline

The remainder of this thesis is structured as follows: First, we will evaluate the broad range of state-of-the-art approaches to ontology matching and give a short account of some representative tools (Chapter 2). Based on this evaluation, Chapter 3 states our main design goals for a flexible ontology matching methodology and sketches out the concepts for its realization. The implementation and integration of the proposed methodology is discussed in Chapter 4. The prototype will then be tested against a benchmark of ontology matching tasks in order to evaluate the quality and applicability of our methodology. The conduct and results of this evaluation are presented in Chapter 5. Chapter 6 concludes the thesis and suggests directions for further research.
2 State of the Art in Ontology Matching

The problem of ontology matching has become prominent in the last decade as the potential of ontologies and their utilization within heterogeneous, distributed environments have attracted attention in various application areas. Apart from leveraging insights from other research areas such as machine learning and schema matching, we also see an effort of researchers to observe the specific characteristics of ontologies and incorporate them in new approaches dedicated to the ontology matching task. A comprehensive survey of research activities that mark the current state of the art in ontology matching and various closely related areas has been presented in [KS05].

In this chapter, we will first distinguish the ontology matching problem against the more “traditional” task of schema matching (Section 2.1). Section 2.2 refers to some general issues that arise when we consider different scenarios in which ontology matching might be applied. After that, we discuss more closely the different aspects of heterogeneity that can arise between two ontologies (Section 2.3). Section 2.4 discusses the special issue of heterogeneity between ontology languages.

The main part of this chapter provides an overview of different aspects of state-of-the-art approaches to ontology matching. First, Section 2.5 elaborates on the broad range of input information we can use to discover mappings, as well as possible representations of the produced mappings, that is, the output of the matching process. In Section 2.6 we then discuss the different algorithms used for ontology matching. We assign the discussed techniques to different classes of matching approaches based on existing classifications. While this evaluation is not intended to be exhaustive, we aim to gain and provide a good insight into various aspects of current ontology matching research. Furthermore, we will present some representative matching systems in more detail in Section 2.7. We conclude this evaluation by discussing what we regard as the major merits and drawbacks of selected approaches in Section 2.8. On this basis, we identify approaches that we deem valuable contributions to our ontology matching methodology.

2.1 Schema Matching vs. Ontology Matching

As mentioned before, the problem of matching ontologies is closely related to that of matching database schemata, which has been an active field of research for quite some time in different application areas such as data integration and semantic query processing. Database schemata and ontologies exhibit many similarities in their definition of a domain vocabulary, the modelling of concepts and their relations, the construction of hierarchical relationships, and the definition of constraints on the defined entities. With those similarities at hand, ontology matching approaches often build upon insights from the database community. A wealth of such approaches has been developed. Comprehensive surveys are provided, for example, in [RB01, SE05].

Although the tasks of ontology matching and schema matching seem to be very similar, the two problems should not be regarded as equivalent. There exist a number of aspects which clearly distinguish database schemata from ontologies [SE05, UG04, NK04]. Often, those differences stem from the differing intended use of schemata and ontologies. In particular, database schemata are conceived in the first place as a means to store and structure data, by building up hierarchical structures and applying constraints on the data. Consequently, the matching of database schemata is concerned with finding correspondences between schema elements so as to integrate linguistically and structurally heterogeneous data sources. Ontologies, on the other hand, are conceived as logical systems describing knowledge (semantics), allowing the application of reasoning techniques on this knowledge. Consequently, ontology matching
is applied to discover semantically similar entities. To provide for a clear conceptualization of the two technologies, we list below the main factors discriminating them.

**Explicit semantics** Ontologies explicitly ascribe semantics to the defined entities and the relations between them. Although such semantics might be implicit in the design of database schemata, they are not included in the final schema realization. In the case of ontologies, a reasoner can be used to infer implicit knowledge from the explicitly and formally defined semantics.

**Relations, constraints and their intended purpose** Database schemata use only hierarchical relations, which define the structure of data. Ontologies, on the other hand, offer a broad range of constructs to convey semantics, such as the specification of concept hierarchies, the declaration of properties, equivalence and disjointness of classes, or local class restrictions. In database schemata we define constraints primarily to enable integrity testing and query optimization. Conversely, the constraints expressed in ontological axioms are explicitly intended to enable automatic reasoning.

**Purpose of use and development process** Even when ontologies and schemata at times exhibit no tangible difference — consider, for example, an ontology that models a mere taxonomy — they usually do differ in their purpose of use. Ontologies are by definition built with the objective of sharing knowledge about a certain domain by providing a common description of the meaning of terms used in that domain. Therefore, their development is essentially distributed, aiming at a maximum of extensibility, reusability, and interoperability. To that end, apart from committing to a common domain description from the start, we can specify ontologies as extensions of other ontologies defined on a higher level, committing to the concept definitions in those high-level ontologies. We resume this aspect below. Database schemata, on the other hand, are usually developed with a particular application case in mind to model the structure of data instances. The aspect of interoperability is therefore usually considered only after the need for integration between two data sources arises.

**Ontology Abstraction Levels as an Approach to Interoperability** As discussed above, ontologies have been promoted primarily as a means to provide interoperability on basis of a commonly agreed-on domain vocabulary. However, it soon became evident that in practice there will hardly ever exist such a common vocabulary to which all users of a domain commit. For example, when developers want to use an ontology in a certain context, they often find they require some context-specific concept description. That is, they need a concept that is modelled with respect to their particular application context, even when some relatively similar concept in the same domain might already exist.

In order to tackle the heterogeneity problem that thus arises, researchers have come up with the idea of providing developers with high-level ontologies. Such ontologies capture the meaning of concepts very abstractly and can be extended by more concrete concepts that fit domain- and application-specific needs. This results in a classification of ontologies with respect to their degree of abstraction. With increasing concreteness of concepts, we can roughly distinguish upper-level ontologies, defining concepts such as Entity and Relation common across all (most) possible domains; core ontologies, which capture such concepts that are valid for some related domains of discourse; and finally, domain ontologies and application ontologies, which model concepts that are specific to a particular domain or application, respectively. Examples for upper-level ontologies are the Suggested Upper Merge Ontology (SUMO) [NP01, SUO03], which has been created by the IEEE Standard Upper Ontology Working Group [SUO03] to enable data interoperability and more effective information retrieval, and the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [Lab07]. Another example for a high-level ontology is the Process Specification Language (PSL) [GM03], which can be thought of as a core ontology defining concepts of process execution in manufacturing and business.

In addition to the abstract high-level ontologies and the concrete descriptions of low-level, task specific ontologies, mid-level ontologies describe concepts of a domain on a medium level of abstraction. They
are intended to bridge between high-level and low-level ontologies. Such a mid-level ontology could, for example, be provided by a company as a common business vocabulary to their various customers, who could in turn extend it with (proprietary) domain- and application-specific concept definitions. The described notion of specifying abstract concepts and extending them with more concrete concept descriptions is very reminiscent of the concept of generalization and specialization applied in object-oriented programming. Furthermore, in both cases the approach stems from the desire for better shareability and reusability of developed components and knowledge. Above, we highlighted that this focus on reuse is an important aspect distinguishing ontologies from database schemata. In the context of ontology matching this aspect can play an important role, since the information provided by high-level ontologies can facilitate the discovery of mappings. We will get back to this issue in Section 2.6.1.

2.2 Ontology Matching in Different Integration Scenarios

We have mentioned above that ontology matching plays a central role for different tasks of ontology reconciliation. Depending on the application context, there exists a variety of scenarios where ontology matching can be applied. We can distinguish those scenarios with respect to different criteria such as the required degree of automation, the time at which the matching shall be performed, or the number and size of the involved ontologies. When an ontology matching approach shall be applied in a particular integration scenario, it has to satisfy different (non-functional) requirements. Below, we briefly discuss some important issues that can arise in this context.

Static and Dynamic Matching  Depending on the application context, ontology matching might be required at different points of time. In most cases, the matching process is conducted statically, that is, at design time. The resulting mappings are then applied in different run-time processes. This is, for example, true for the scenario of catalogue integration. However, there exist also scenarios where the matching must be executed at run time, such as the scenario of agent communication described in Section 1.2, where we most probably need to match between ontologies as communication takes place. Design and run time problems incur different resource requirements on matching approaches. A system that must deliver reasonable results in real time will certainly have to adhere to more rigorous resource restrictions than one that is intended for design time matching. Consequently, such a system must take a different approach to the matching task.

Degree of Automation  The point of time at which a matching is done also influences to what degree a user can interact in the matching process. Apart from that, there are other aspects that influence how much support a user can provide in the matching process, such as the amount of development time available and the expertise of the involved user. To date, most matching approaches work as semi-automatic approaches, that is, a user is responsible for configuration of the process (pre-matching) and correction of the matching results (post-matching). Fully automated matching is not considered realistic, yet various degrees of automation can currently be observed in different tools.

Size of the concerned ontologies  Since the task of ontology matching is concerned with the discovery of correspondences between ontology elements, it is apparent that the computation costs increase (exponentially) with the size of the ontologies. This issue must be taken into consideration when a matching approach shall be applied to large-size ontologies, which can consist of thousands of elements. In such cases, the applied approach should take measures to reduce computation costs, possibly trading off matching quality.
Mapping Scenario Architectures There exist different high-level integration architectures that account for the different characteristics of matching tasks and the involved ontologies. Similar architectures have been devised for different application fields, for example, in the field of information integration [WVV+01]. In the following, we distinguish architectures for local matching, matching based on a global common ontology and cluster-based matching.

- **Local matching between ontologies** In the first architectural approach, matching is done between each pair of ontologies that need to be reconciled in a given scenario, as illustrated by the arrows in Figure 2.1. This local matching approach is very simple and flexible due to its on-need basis. However, it is obvious that the number of possible (and thus potentially produced) ontology alignments is very high — \( O(n^2) \) alignments are produced for \( n \) ontologies in the worst case. Therefore, this architecture scales very poorly when many ontologies are involved.

- **Matching to a global common ontology** A converse approach is to establish a single common ontology against which all ontologies are matched, as illustrated in Figure 2.2. This common ontology can then be used as a pivot point for matchings between the single ontologies, represented in the figure by dashed arrows. Clearly, this approach reduces the number of required mappings to at most \( O(n) \), and therefore is more scalable. However, it must be noted that design and maintenance of the common ontology incurs a great cost. In contrast to the first approach, acquired mappings can also no longer be maintained locally, but need to be updated whenever the common ontology is altered.

The second approach, where a common ontology serves as “interlingua” between local ontologies, is applicable in more static scenarios, such as in the scenario of catalogue integration. In a dynamic scenario on the other hand, the maintenance of a core ontology would incur disproportional costs. Therefore, the local matching architecture seems better suited to a dynamic environment where high flexibility must be maintained and the involved ontologies are usually small in size.

- **Cluster-based matching** An approach that constitutes a trade-off between the two aforementioned architectures is presented in [VT99]. The authors describe how a number of common ontologies is applied to form several clusters in the overall ontology space, as illustrated in Figure 2.3. While the individual ontologies are matched to their respective cluster’s common ontology, the common ontologies are matched to each other to allow for information interoperability between the clusters. The matching between individual ontologies (represented by dashed arrows in Figure 2.3) is then performed based on the intra- and inter-cluster alignments. This architecture combines the advantages of the first two approaches in that it scales well but also allows for local maintainability and flexibility.
2.3 Categorization of Ontology Mismatches

The goal of ontology matching is to find semantically corresponding pairs of elements in two heterogeneous ontologies in order to be able to bridge the heterogeneity. In this section, we discuss what the term heterogeneity means precisely in the context of ontology matching.

In fact, we can state that there exist many different forms of heterogeneity between ontologies. In an overview devoted to this issue, Klein [Kle01] provides a classification of such types of heterogeneity, so-called ontology mismatches, building upon findings in [VJCS97] and other work. Following his view, we can distinguish between mismatches at the language (metamodel) level and at the ontology (model) level. The first class refers to such mismatches that occur when we want to combine ontologies represented in different ontology languages. This concerns the syntax, the logical representation and semantics of language constructs, and, most importantly, the expressivity of the language. We get back to the issue of language-level mismatches in Section 2.4.

Mismatches at the ontology level, on the other hand, occur when ontologies refer to the same or overlapping domains of discourse but the description of the domains differs in some way. This is the class of mismatches this thesis focuses on. We will briefly discuss the different sorts of ontology-level mismatches below and provide additional information throughout the thesis whenever this is necessary.

At the highest level, we distinguish mismatches in the conceptualization and the explication of a considered domain.

**Conceptualization mismatches** The first class of ontology-level mismatches comprises such differences that occur in the conceptualization of the domain. That is, the semantics two ontologies describe for a certain domain and its concepts can be different. This includes the following aspects:

- **Scope of concepts**: The set of instances a concept covers in one ontology can differ from the set of instances a seemingly similar concept has in another ontology.

- **Coverage**: Two ontologies might cover different portions of (a domain of) the world. Whereas one ontology includes a certain concept in its description of a domain, another might ignore it. For example, while a bibliographic ontology includes various types of publications such as Book, Article, and Report, a bookshop ontology might include only concepts such as Book and Monograph.

- **Granularity**: The degree of detail to which a certain concept is modelled can vary between ontologies, depending on what is regarded as relevant in the respective ontology. For example, a bibliographic ontology models various types of publications, whereas another (more general-purpose) ontology might only model a generic concept Publication.

- **View point**: The perspective from which the ontology is defined can influence the definition of entities. The meaning of concepts can, for example, vary depending on the place and time at which an ontology is defined.

**Explication mismatches** This class comprises such mismatches that occur in the specification of the conceptualization, that is to say, in the way we describe the semantics we want to represent (whereas the type of conceptualization mismatches considered the intended semantics itself). The class of explication mismatches comprises the following aspects:

- **Style of modelling**: The paradigms and description conventions used for the modelling of concepts can differ. For example, the level at which concept distinctions are manifested in a class hierarchy can be chosen higher or lower depending on the intended use of the ontology.
Terminological mismatches: Differences in the naming of concepts are arguably the most commonly observed. Concepts from two ontologies that correspond to each other can be represented by different names (synonymy). Conversely, similar terms which are used in two ontologies can refer to different concepts in their ontology’s respective context (polysemy). Another form of terminological mismatch occurs in the use of different languages and syntactic variations of words, for example, when different naming conventions are used.

Encoding: The formats that are used to encode values can differ. For example, date and time strings can follow different patterns.

An additional source of heterogeneity, which is often neglected in the different works on this topic, is the issue of pragmatics. The intended use of an ontology can influence to a large degree the way it is constructed. Even when two entities seem to be related to each other, they might indeed have no reasonable relationship when taking into account the intended purpose of the ontologies.

Concluding, we observe a broad range of possible mismatches between ontologies. To bridge such mismatches, it is necessary to identify those parts of the element descriptions where no or only minor mismatch occurs. Having found such aspects in which the ontologies “overlap”, we can apply different approaches that leverage this information to find correspondences between elements.

While the classes of conceptualization and pragmatic mismatches constitute very advanced problems, which are not considered in most current approaches, the explication mismatches have been addressed at length. In Section 2.6 we elaborate on a variety of dedicated ontology matching approaches.

2.4 Ontology Languages and Language-Level Mismatches

As mentioned above, heterogeneity between ontologies can not only occur in the modelling of ontologies, but also at the language level, that is, because two ontologies are expressed in different ontology languages such as RDF Schema [BG04], OWL [W3C04a], F-Logic [KLW95], or Loom [Loo07]. According to [Kle01], we can observe the following four types of language-level mismatch, relating to different aspects of a language:

Syntax Different ontology languages use different syntaxes to represent similar concepts. This mismatch can be solved easily in most cases, for example, using rewrite mechanisms. However, this mismatch often comes coupled with one or several of the following, in which case it is harder to resolve.

Logical representation Similar logical notions are often represented by different language constructs, while the semantics of the representations are equivalent. We can solve this mismatch by providing translation rules to convert one representation to the other.

Semantics of primitives This mismatch is much harder to solve than the previous ones. It refers to seemingly similar language constructs (i.e., constructs which are conceptually similar, or even have the same syntax), which however differ in their semantics.

Language expressivity The most critical mismatch between languages is that of expressivity, that is, the things that can be expressed in a given language. This mismatch can not be resolved completely, and therefore poses the hardest problems when relating different language representations in the context of ontology management, and particularly so in the context of ontology matching.

In the remainder of this section, we briefly discuss two prominent knowledge representation languages, which are currently used for ontology specifications: The W3C standard OWL and the frame-based, object-oriented F-Logic. We give a brief overview of the characteristics of the two languages, as well as
2.4. ONTOLOGY LANGUAGES AND LANGUAGE-LEVEL MISMATCHES

the underlying logics, and discuss them in the context of language level mismatches. A comprehensive discussion of available ontology specification languages is out of the scope of this thesis. A study that evaluates different languages for ontology representation and gives a detailed insight about the kind of mismatches that can occur at the level of language expressivity can be found in [CGP00].

2.4.1 OWL — A DL-based Ontology Language

The Web Ontology Language (OWL) has been designed by the W3C Web-Ontology Working Group [W3C04b] specifically to enable the representation of ontological knowledge in the context of the emerging Semantic Web. Its semantics can be translated into Description Logics (DL), and implemented DL reasoners can be used for inference over OWL ontologies.

Description Logics  DL (see, e.g., [BHS04]) are a decidable subset of first-order logic. They have a formal, logic-based semantics and can be used to describe the knowledge of a domain in a structured way. The notions of a domain are described by concept descriptions, which are built from atomic concepts (unary predicates) and roles (binary predicates), using different constructors such as boolean constructors or restriction constructors. This description formalism is accompanied with a terminological formalism to introduce names for the above-mentioned complex concept definitions. Furthermore, an assertional formalism is used to describe properties of individuals, for example, stating that an individual belongs to a particular concept.

Inference algorithms can work on the DL semantics to infer implicit knowledge from the explicitly represented knowledge in the knowledge base. Using a subsumption algorithm, a reasoner can infer subconcept-superconcept relationships from the given descriptions. Furthermore, using the instance algorithm, it can determine whether an individual belongs to a particular concept description. Finally, the reasoner can check whether the assertions and terminological axioms of a knowledge base are non-contradictory, thus checking the consistency of the knowledge base. In order for such inference problems to perform in reasonable time, the expressivity of the DL must be restricted, while it must remain powerful enough to express the notions of a domain.

The Web Ontology Language  In the design of OWL, developers have aimed to find such a compromise between expressivity and complexity. OWL builds upon RDF and RDF Schema, using RDF syntax and most of RDF Schema modelling primitives. The language design is based on the DL \textbf{SHIQ}, which provides for decidable reasoning and is expressible enough to be applied as ontology language [BHS04]. For more information on the \textbf{SHIQ} language features and reasoning, see, for example, [BHS04, HPSvH03, HST00].

There are three flavors of OWL: OWL Lite, OWL DL and OWL full, each one being an extension of its predecessor, thus accounting for different needs for expressivity and reasoning efficiency. While OWL Lite highly restricts the language expressivity and is mainly applicable to define simple concept taxonomies, OWL DL is very expressive and can be used to define complex axioms. It maintains decidability of the logic by restricting the way in which constructors can be used. OWL full trades off this efficiency of reasoning for full upward compatibility with RDF, including the possibility to change the meaning of predefined primitives. In this work, the focus is on ontologies described in OWL DL.

Analogue to the DL description formalism, an OWL ontology describes the domain in terms of classes, which are either simple named concepts or built up in turn from other classes and properties using the available constructors. It contains a set of axioms, which assert, for example, subsumption or equivalence relationships between classes and properties.

We do not want to evaluate the language in detail here. Rather, we will give more detailed information in the further course of this thesis, where this is necessary. For the complete language specification and
2. STATE OF THE ART IN ONTOLOGY MATCHING

2.4.2 F-Logic — A Frame-based Ontology Language

F-Logic is a frame-based language, which takes an object-oriented approach to knowledge representation. Correspondingly, objects are the basic constructs in F-Logic. They model real world entities and can be accessed through their unique object name. This is in contrast to the view taken in DL, where instances are classified based on class and property descriptions, and do not necessarily have a unique name.

Object and variable names (id-terms) form the basic syntactic elements in F-Logic. Complex id-terms can be built by applying function symbols, which in turn take other id-terms as arguments. Information about class hierarchies, class membership and relationships between objects is expressed using F-atoms. More precisely, subclass-F-atoms and is-a-F-atoms are used to define subclass relationships and class membership, respectively. Relations between objects are expressed as method applications on those objects in the form of data-F-atoms, which relate a host object, a method, and a result object, possibly applying a parameter list. We can furthermore specify explicit typing constraints using signature-F-atoms, which declare methods on classes and restrict the applicable parameter and result types. Another salient feature of F-Logic is the definition of rules, which can be used to derive new information from a given object base. Such rules consist of a precondition (the rule body) and a conclusion (the rule head). Whenever the precondition is satisfied, then so is the conclusion. This rule mechanism facilitates the specification of complex generic knowledge, thus boosting the expressivity of the language.

F-Logic uses a higher-order syntax. However, its semantics can be expressed equivalently using first-order predicate expressions. In contrast to OWL DL, F-Logic is not decidable. Still, implemented inference systems such as Ontobroker[DEFS98] and Flora [YK00] are sufficiently efficient for run-time application [AL04].

Concluding, F-Logic combines important benefits of object-oriented systems, such as typing and non-monotonic inheritance, with well-defined logical semantics and a very high expressivity. For an overview of the syntax and semantics, as well as information about implementations of F-Logic, see, for example, [AL04, KLW95].

2.4.3 Mismatches between OWL and F-Logic

Resuming the issue of language-level mismatches, we now briefly compare the two languages described above, singling out some of their differences with the aim to exemplify the discussed language-level mismatch types.

Syntax The syntaxes of OWL and F-Logic are entirely different. While OWL uses a very verbose XML syntax, designed to be human-readable, F-Logic applies a rather compact, logics-based syntax. For example, to express a subclass relationship between classes $A$ and $B$ in F-Logic, we write $A : B$. To express the same notion in OWL, we would write:

```xml
<owl:Class rdf:ID="A">
  <rdfs:subClassOf rdf:resource="#B"/>
</owl:Class>
```

Logical representation The two languages provide many similar features, but represent the intended semantics through different constructs. One example for such a representation mismatch is how property characteristics are specified in F-Logic and OWL, respectively. For example, to state that a
property isPartOf is transitive, we can define a rule in F-Logic as follows:

\[ \text{FORALL } X, Y, Z \ X[\text{isPartOf} \rightarrow Y] \leftarrow X[\text{isPartOf} \rightarrow Z] \text{ AND } Z[\text{isPartOf} \rightarrow Y]. \]

To express the same semantics in OWL, we would use the axiom `owl:TransitiveProperty`:

\[ <\text{owl:ObjectProperty} \text{ rdf:ID="isPartOf"}> <\text{rdfs:domain} \text{ rdf:resource="&owl;\;TransitiveProperty"/> \text{owl:ObjectProperty}> \]

**Semantics of primitives**

The semantics of many constructs used in the two languages are similar at first glance, but indeed exhibit more or less subtle differences when considering the logical foundations that underlie the languages.

Reasoning in F-Logic and OWL is based on fundamentally different assumptions about the nature of the described world. F-Logic applies the *Closed World Assumption (CWA)*. This means that if we do not know that something is true, that is, a certain fact has not been expressed explicitly, we assume that it is false. F-Logic reasoning therefore is non-monotonic, since addition of new facts can invalidate old inferences. In contrast, OWL as a DL follows the *Open World Assumption (OWA)*. Therefore, reasoning over OWL ontologies is monotonic, that is, the addition of new assertions to the model can only add new inferences, not invalidate old ones. Based on this major difference in the underlying logics, the two languages exhibit many mismatches on the level of semantics.

- **Monotonic vs. non-monotonic inheritance**
  The semantics of the inheritance construct of OWL DL and that of F-Logic is different. OWL applies only monotonic inheritance, while F-Logic uses non-monotonic inheritance. In monotonic inheritance, a subclass can add new properties or property values to the inherited features. In non-monotonic inheritance, we can also change the inherited features, that is, replace or cancel an inherited property value.

- **Unique names assumption**: F-Logic assumes unique names for all its elements, in contrast to OWL DL. If two elements have different names in OWL, we cannot infer that the elements represented by those names are different as well.

- **Typing**: Specifying the range of a property implies different semantics in F-Logic and OWL. F-Logic allows type restrictions by means of method signatures. Conversely, when specifying the range of a property in OWL, this does not imply strict typing. Instead, a reasoner might use the information to infer new knowledge.

**Language expressivity**

The issues presented above apparently also imply differences in the expressivity of the two languages. For example, we can not express restrictive typing information in OWL in contrast to F-Logic, where we can use method signatures to do so. In general, F-Logic has a higher expressivity than OWL DL. A very powerful feature of F-Logic which has no equivalent in OWL DL is the rule mechanism described above. Also, F-Logic allows to treat classes as instances of other classes, which is ruled out by the language restrictions of OWL DL.

**2.4.4 Conclusion**

Concluding the (non-exhaustive) considerations on heterogeneity between OWL and F-Logic, we see that the languages exhibit various mismatches of different types. This diversity of mismatches makes the problem of *language normalization* a highly complex issue. This is true above all for the differences observed in the semantics and expressivity of the languages. Even when no inference mechanism is included in the matching process, we still have to ensure that the discovered mappings are semantically sound. To provide a basis for language normalization in ontology matching, the discussed issues must be investigated thoroughly for each language we wish to support.
2.5 The Matching Process

The remainder of this chapter is devoted to a detailed evaluation of ontology matching approaches that address ontology-level mismatches. In this section, we first discuss the general ontology matching process. We give a high-level description of the different process phases we can observe in most ontology matching systems. Then, we discuss the different types of input that can be utilized for mapping discovery, as well as various possible forms to represent the produced mappings.

2.5.1 A Generic Matching Process

The matching of ontologies follows a similar generic process model in most mapping tools documented in the literature. MAFRA [SR03], a conceptual MAppling FRAmework for distributed ontologies, identifies five stages within the matching process. Those include a phase for mapping execution, that is, the transformation of elements from the source into elements of the target ontology according to the obtained mappings. Apart from this transformation step (which, by itself, is not a part of the matching process as defined in Section 1.4), most current systems realize the described phases to some degree.

1. **The lift & normalization** step includes the normalization of different ontology specification languages to the same language or internal representation. Furthermore, basic linguistic processing such as the expansion of abbreviations is conducted at this stage.

2. **Computation of similarities** is performed for all element pairs from the involved ontologies. This computation can be based on various features of the ontology elements, such as linguistic features and hierarchic information.

3. In the phase of **semantic bridging** the system determines, based on the previously computed similarities, correspondences between elements so that every element in the source ontology is related to the most similar element in the target ontology. In this phase, MAFRA also accounts for additional domain knowledge or prior mapping results that might be introduced into the process.

4. The next step is the **execution** of the generated semantic bridges to transform source ontology elements into elements of the target ontology.

5. Finally, in the **postprocessing** stage, the obtained mappings and transformations are reviewed and refined, possibly combining manual labour and automatic support.

The phase of language normalization refers to the problem of language-level mismatches discussed in the previous section. In the remainder of this chapter, we will mainly address the phases of similarity computation and semantic bridging, discussing the input and output of the matching process (Section 2.5.2 and Section 2.5.3), as well as the applied algorithms for mapping discovery (Section 2.6).

2.5.2 Matching Input: Information Utilized in the Matching Process

As a basis for an evaluation of different state-of-the-art matching approaches, this section discusses the range of input features that can help to discover mappings between ontology elements. Here, we go along the lines of the categorizations of matching approaches in [RB01, SE05], focusing on the input aspect.

We can group input information along two dimensions:
2.5. THE MATCHING PROCESS

**Type of input:** We distinguish between *linguistic, taxonomic, semantic,* and *model* input information.\(^1\) In addition, we consider possible *pragmatic* input information that might influence the matching.

**Origin of input:** Input information can be either *internal,* that is, such features as are inherent in the definition of the two input ontologies (and their instances\(^2\)) or *external,* that is, such information as comes from additional sources.

Table 2.1 groups the sorts of input along the described dimensions. Below, we briefly discuss the different input types. We will refer back to them in context when discussing various matching approaches in Section 2.6.

<table>
<thead>
<tr>
<th></th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linguistic</td>
<td>Entity labels, comments</td>
<td>Lexica (e.g., WordNet), thesauri, abbreviations</td>
</tr>
<tr>
<td>Taxonomic</td>
<td>Concept hierarchy</td>
<td>(Referenced) Upper-Level ontologies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prior alignments and similarities</td>
</tr>
<tr>
<td>Semantic</td>
<td>Element descriptions, Instances</td>
<td>(Referenced) Upper-Level ontologies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prior alignments and similarities, Annotated documents</td>
</tr>
<tr>
<td>Model</td>
<td>Model-theoretic semantics</td>
<td>Axiomatic background knowledge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Referenced) Upper-Level ontologies</td>
</tr>
<tr>
<td>Pragmatic</td>
<td>Version and time stamp</td>
<td>User input (e.g., process parameters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information about users, communities, networks</td>
</tr>
</tbody>
</table>

Table 2.1: Categorization of input data

**Linguistic** information comes in the form of entity labels and comments that intuitively describe ontology elements. Furthermore, we can use linguistic data to derive related external linguistic information by exploiting accessible lexica, (domain-specific) thesauri, or abbreviation indices.

**Taxonomic** information comprises information about the concept hierarchy of an ontology. We can exploit various external information such as upper-level ontologies, previously obtained mappings, and similarities between ontology fragments to complement the hierarchic information from the ontologies themselves.

**Semantic** information comprises the various information parts we can extract from ontology element descriptions, such as property or restriction information. Again, we can use external sources to complement the semantic information from the ontologies. Furthermore, we can exploit the instances of ontology elements, since they represent real-world content of a concept, thus modelling its semantics (see, for example, [DDH01]). When no instances are available, we can provide them in the form of documents annotated with concepts from the ontology.

**Model** information denotes such input that can be exploited by deductive methods. For example, we can leverage the DL semantics of OWL DL ontologies. Alternatively, we can encode the ontology

---

\(^1\) In [SE05], *semantic* input is constrained to such information that can be handled based on its semantic interpretation and thus be exploited by deductive methods, such as DL reasoning. We depart from this categorization, introducing an additional category *model.* This is because we denote as *semantic* the different semantic features that are described in ontologies, such as property or restriction information. Taxonomic information is thus a special sort of semantic information, which considers the concept hierarchy of the ontology.

\(^2\) With the establishment of OWL as a de facto standard for ontology representation, instances are often considered as part of the ontology. Strictly speaking, they are not. We refer to them as internal input information because they can be extracted directly from OWL ontologies.
matching problem (comprised of many element pair mapping problems) in a set of propositional formula and use axiomatic background knowledge [SE05]. Upper ontologies can also be lever-
aged, complementing the model information of the input ontologies.

Pragmatic information is related with the task-specific pragmatics, that is, somehow determined by context-dependent practical issues, such as the intended use of the alignment, or the user’s assumption about the nature of the matching task. For example, a user can parameterize the matching process according to the mismatch he considers important in the respective task. Social network information (e.g., membership in a community) can also play a role as to how a process is configured or which mappings are considered plausible.

2.5.3 Matching Output: Representation of Discovered Mappings

As output of an ontology mapping process, we obtain a set of mappings between elements of the input ontologies. As stated in the introduction of this thesis, those mappings are of little use by themselves, but we use them as basis for further steps, such as translation between ontologies or ontology merging. In order to be able to use an alignment for such tasks, the characteristics of the obtained mappings are relevant.

We can distinguish mappings with respect to the information they provide. As stated in Section 1.4, mappings state correspondences between elements, including an expression specifying the exact relation between the elements and a confidence measure indicating the degree of belief that the mapping holds. Most current systems produce mappings that relate two elements with each other based solely on a confidence measure. Implicitly, they state thereby that the elements are highly similar, or equivalent. Only very few approaches such as CTXMATCH [BSZ04] support the discovery of explicit semantic mappings, such as expressing subsumption relationships [SdM06]. Depending on the expression associated with a mapping, it can be either uni- or bidirectional. For example, when a subconcept relation is specified between two mapped objects, the mapping is unidirectional. Furthermore, mappings can have different cardinalities. They can relate a pair of elements (i.e., describe 1-1 correspondences such as “class A is equivalent to class B”) or relate several elements (i.e., describe 1-n, n-1 or m-n correspondences such as “class A is a union of classes B and C”). Most current research still focuses on 1-1 correspondences.

Another distinction we make is with respect to the way mappings are represented. The required representation depends essentially on the different purpose of the produced mappings, that is, how we want to use them in further processing steps.

Abstract representations for mapping frameworks: Striving for a sound formalization of the entire matching problem, researchers try to capture general characteristics of mappings in a generic, representation-independent form. An example is the representation of mappings as queries and views, which has its origin in the database integration architecture [CGL01]. In [BEE+04], a framework of different mapping types and elements is described. In [CM04], mappings are expressed as instances of a mapping ontology describing their various characteristics. Providing a common vocabulary for the description of mappings, such an ontology could facilitate exchange, maintenance and reuse of ontology mappings between different tools.

Reasoning across reconciled ontologies: One of the major benefits of ontologies over other types of knowledge representation is the ability to reason over the knowledge they express. To support this functionality across multiple (matched) ontologies, the produced mappings must be represented in such a way that reasoners can work on them, that is, in some axiomatic form. To this end, the language Context OWL (C-OWL) [BGvH+03] extends OWL with support for mapping descriptions. This way, C-OWL enables the connection of OWL expressions and distributed reasoning across different ontologies.
2.6. APPROACHES APPLIED FOR DISCOVERY OF MAPPINGS

In [Dou04], a set of bridging axioms describes the mappings between ontologies. A bridging axiom can be regarded as any statement which relates a pair of elements (when regarding 1-1 mappings) or an arbitrary number of elements (when regarding m-n mappings) from two ontologies. The representation of the axioms is in the form of FOL axioms. When combining the bridging axioms with the source ontologies, a reasoner can be applied on this merged model.

Transformation of data between ontologies: The above-mentioned bridging axioms [Dou04] can be applied in various integration tasks. An axiom essentially describes a view from one ontology onto the other, where entities (predicates) of one ontology are defined in terms of entities of the other. On this basis, translation can be carried out between the ontologies.

Another approach is taken in [CM04], where the above-mentioned mapping ontology describes the structure of different mapping types, as well as transformation rules and conditions and effects that apply to such rules. Integration tools can use instances of the ontology (i.e., discovered mappings) for automatic processing.

2.6 Approaches Applied for Discovery of Mappings

In the previous section we discussed possible input information and the representation of the resulting output mappings. Now, we finally flesh out the process of mapping discovery itself. Investigating state-of-the-art systems and theoretical works on ontology matching, we can distinguish a wealth of techniques adopted from various fields, including traditional database and Information Retrieval (IR) techniques, approaches from Natural Language Processing (NLP), machine learning, and reasoning.

In the following, we provide a classification of ontology matching approaches. It is based on previous classifications presented in [RB01, SE05]. We adapt the classification in [RB01], which describes approaches to schema matching, extending it with concepts from [SE05] in order to include ontology-specific aspects. We point out that, due to the manifold aspects that can be considered when investigating ontology matching approaches, various other categorizations are conceivable.

Often, an applied approach can not be classified definitely in any one category of approaches. In fact, virtually all currently promoted systems use a combination of different techniques, or matchers, in order to optimize the matching result.

Integration of matching approaches Following the view taken in [RB01], we first classify ontology matching approaches with respect to how different techniques are integrated in the matching process, as shown in Figure 2.4.

![Ontology Matching Approaches Diagram](image)

Figure 2.4: Integration of matching approaches

First, there are individual matcher approaches, which compute the matching result based on a single
criterion, such as the name of entities. Second, there are combined matcher approaches, which combine different techniques and exploit various features of the ontology to compute an alignment. As stated above, virtually all state-of-the-art systems use some kind of combined approach. Combined matchers can in turn be discerned into hybrid matchers, which compute the matching result by exploiting various criteria at once, and composite matchers. Composite matchers compute the alignment based on a combination of results obtained from various other matchers, which can be individual or combined matchers, in turn. For a composite matcher, we must configure which matchers they shall be composed of, as well as how this composition shall take place. Configuration could be done either manually or (semi-)automatically, depending on the provided configuration facilities.

**Individual matcher approaches** In the remainder of this section, we discuss different individual matcher approaches. The classification of those approaches is illustrated in Figure 2.6. Adopting the criteria used in [RB01], we distinguish the techniques first with respect to whether they are ontology-based or instance-based:

**Ontology-based approaches** consider information on the level of ontology entity descriptions, such as classes and properties.

**Instance-level approaches** exploit information about the instances (the extension) of an ontology to infer correspondences between the entities those instances belong to. If the ontology is not populated with instances, that is, no (not enough) instances are defined for the ontology concepts, such instances might be obtained manually or automatically from external sources.

A second distinction of approaches can be made with respect to the granularity of the matching process. Here, we distinguish between element-level and structure-level approaches.

**Element-level approaches** consider the individual entities of an ontology in isolation, ignoring the relations with other entities. They rely essentially on linguistic information such as concept labels, or constraint information such as the cardinalities of properties.

**Structure-level approaches** consider how entities appear together with other entities in a structure. This structure can be defined by hierarchical relations, but also by other semantic relations, such as the property relation.

As shown in Figure 2.6, the ontology-based element-level approaches comprise string-based, lexical, constraint-based, reference-based, and reuse-based approaches. On the structure-level, we consider graph-based, taxonomy-based, model-based, and reuse-based approaches.

As instance-based approaches we observe on the element-level linguistic and constraint-based approaches. Furthermore, there are probabilistic approaches, which use a probabilistic definition of similarity.

In the following, we discuss each of the above-mentioned approaches in turn. In doing so, we also refer to systems that realize the respective techniques.

### 2.6.1 Ontology-based Approaches

As ontology-based approaches we denote such algorithms that base their computation of similarities on information extracted from the intentional descriptions in the ontology, that is, the descriptions of class and property entities. As stated above, we distinguish between approaches at the element-level and at the structure-level.
2.6. APPROACHES APPLIED FOR DISCOVERY OF MAPPINGS

2.6.1.1 Element-Level Approaches

Element-level techniques consider the ontology elements in isolation and rely essentially on such information as is available in the linguistic description of an element or the constraints defined on an element, such as the datatype and cardinality of properties. The extracted information is utilized either directly, for example, by checking two strings for equality, or by introducing auxiliary information, such as external thesauri. The following approaches can be discerned:

**String-based approaches** compare names and descriptions of entities and on this basis compute a similarity between the entities. Words are considered as sequences of letters, that is, in terms of their syntax. This approach is based on the intuition that entities with syntactically similar names are likely to correspond to each other, that is, represent semantically similar concepts. Obviously, this assumption is sometimes invalid, for example, when syntactically similar words have different senses (polysemy).

In most cases, *language-based techniques* are applied as a preprocessing step for string-based approaches. They are grounded on NLP techniques exploiting morphological characteristics of the words. On this basis, composite words are split into their constituent words (tokenization), those are converted to their basic forms (lemmatization), and function words are discarded (elimination). There exists a great variety of string similarity measures, such as *EditDistance*, *N-Gram*, or measures comparing the *prefixes* or *suffixes* of words. For an evaluation of such algorithms, see, for example, [CRF03]. String-based approaches are applied in some form in virtually all matching systems.

**Lexical approaches** use lexical auxiliary resources, comparing entity names based on linguistic relations between them. Here, words are considered with respect to their linguistic semantics rather than as a mere string of letters. Lexica or (domain specific) thesauri can be used in order to evaluate, for example, synonym and hypernym (superconcept) relationships between words. The lexical database WordNet [CSL06, Mil93] can be used as a resource of lexical information. It distinguishes for each contained word the different *senses* that are associated with it and groups words with similar senses into *synsets* (sets of synonyms), which are linked by different relations. On this basis, the similarity of two concepts can be computed based on links between their synsets. This approach is used, for example, in HCOME [KV04], CTXMATCH [BSZ04], and iMapper [SG05].

![Diagram of Individual matcher approaches](image-url)
Constraint-based approaches exploit information about internal constraints in the definition of entities, such as the datatype and the cardinality of a property. Datatypes can be compared based on a (subjective) rating of how “close” they are, using predefined compatibility information. For example, the datatype `date` could be considered closer to the datatype `string` than to the datatype `float`. Taking a more advanced approach, we can compute datatype similarity based on set-theoretic comparison [SE05]. Other constraints, such as the cardinalities of properties, can also be compared, using either some specified rule or simple numeric similarity measures [EV04]. For example, we might regard a set of 5-10 employees as more similar to a set of 10 people than a set of 2 employees.

Reference-based approaches exploit references to a high-level ontology to find correspondences between elements that are extensions to the high-level concepts. In Section 2.1 we have described the notion of extending ontologies from a higher-level ontology. When developers in different contexts build their ontologies as sound extensions of the same ontology, we can exploit it as a common grounding vocabulary in the matching process. This can facilitate the discovery of mappings. If, for example, two classes have different names but are specified as the extension of the same top-level concept, we can consider them similar.

As a possible scenario for this case, imagine a company providing a core ontology to their clients, therein describing business objects such as `Employee`, `Invoice`, `Product` and the relations between them. Clients could build their own ontologies by extending this core ontology in an application specific way that suits their particular needs. They would then be able to match between their respective ontologies on basis of the common ontology.

In Section 2.1, we introduced PSL as an example for a core ontology. In [GK05], PSL is used as an application independent interchange ontology by defining mappings between application-specific ontologies and PSL and then using those mappings to match between the application-specific ontologies.

We also observe endeavours to generally facilitate the use of upper-level ontologies in the matching process, even if the concerned ontologies do not refer to a common ontology. To that end, WordNet has been aligned with SUMO [NP03], that is, WordNet synsets were mapped to corresponding SUMO concepts. Those mappings can be used in ontology matching, for example, to find relations of entity names from the input ontologies to SUMO concepts. This way, SUMO can be used as a common ontology in the matching process, even when the ontologies have not been developed as extensions of SUMO.

Reuse-based approaches exploit knowledge about previously computed alignments, which have been stored in a repository. The idea behind this approach is that ontologies are often similar to other ontologies previously matched. If, for example, we consider two elements $e_1$ and $e_2$, and the alignment repository contains mappings between $e_1$ and $e_3$ and between $e_3$ and $e_2$, then we can deduce a mapping between $e_1$ and $e_2$. We can reuse either the alignments of entire ontologies, or of parts thereof (fragments). This approach was introduced in [RB01] and implemented, for example, in [Do05].

2.6.1.2 Structure-Level Approaches

Structure-level approaches consider how ontology elements appear in a structure defined by some ontological relation (e.g., hierarchical or property relations) and compute similarities based on adjacent elements in such a structure.

Graph-based approaches represent the input ontologies in the form of graph structures, where nodes and edges model the ontology entities and the relations between them, respectively. In
2.6. APPROACHES APPLIED FOR DISCOVERY OF MAPPINGS

In the case of simple taxonomic ontologies, which only contain classes and their sub and super-class relations, the edges of such a graph represent the subsumption relationships between the concepts of the ontology, which are modelled as nodes. However, graphs can also be used to model the whole spectrum of semantic information expressed in ontologies. This approach is, for example, taken in [EV04].

Considering such a graph in the ontology matching process, entities are compared with respect to their position in the graph, respectively the adjacent elements in the graph. This approach is grounded in the idea that if two nodes in the graph are similar, then probably their adjacent nodes and edges are as well similar, and vice versa. On this basis, different methods can be applied on the graph.

In [MGMR02], the authors present a dedicated graph matching technique, called *Similarity Flooding*. Here, the goal is to minimize the dissimilarity of matched objects in the graph, that is, to solve an optimization problem, using a fix-point algorithm. Furthermore, there is a broad range of approaches that consider the similarity of nodes in the graph based, for example, on the similarity of their children or parent classes, properties, or otherwise related nodes. This amounts to a propagation of similarities from one element to another in the graph. Some systems such as [ES04, MNJ04] use manually defined mapping rules to encode such graph-based similarity information (e.g., the rule “If the children of concept \( A \) are similar to the children of concept \( B \) then concept \( A \) is similar to concept \( B \)”). Another approach, which relies solely on the structural information in ontologies (without regarding the linguistic information in the nodes) is GMO [HJQW05]. This algorithm represents ontologies as RDF bipartite graphs and measures their structural similarity.

**Taxonomy-based approaches** are close to the graph-based approaches, using similar algorithms. However, they only consider taxonomic relations, that is, subclass-supersclass relationships between concepts. The motivation is analogue to the one described above, namely, that elements connected through generalization are relatively similar, and therefore their neighbouring elements in the taxonomy are likely to be similar as well. Most systems exploit such taxonomy-based approaches to some degree. **Bounded path matching**, as applied in [NM02], exploits the taxonomy structure by extracting the paths between concepts as defined by the hierarchical relations and comparing them based on individual terms and the position of terms within the path.

**Model-based approaches** work on the basis of the model-theoretic semantics of the input. In order to apply inference on different kinds of input information, we must first make this information available in some form of logical representation. In the case of ontologies in OWL DL, we can apply a DL reasoner on the complete set of entities from both ontologies [SE05]. By applying subsumption on each pair of entities from that set, a reasoner can discover and thus map those entity pairs that have the same interpretation.

Another approach is to encode all potential mapping pairs of a matching problem into a set of propositional formulas, which can be checked for validity by a SAT prover. Furthermore, we can introduce manually defined axiomatic rules encoding background knowledge. In [BSZ04], the authors describe an approach that uses logical formulas to encode all available (linguistic, structural and domain specific) information about the ontology elements.

**Reuse-based approaches** at the structure-level are based on a repository that stores structures (fragments) of ontologies and their pairwise similarities [SE05]. The idea here is that some computationally cheap algorithm can be employed for computing the similarities between fragments, and then to compare only such fragments in full detail where the similarity between the fragments exceeds a certain threshold. This approach is described and implemented in [Do05].
2.6.2 Instance-based approaches

Instance-based approaches are grounded in the general idea that the semantics of ontology entities to be matched are modelled to a high degree by their instances, which are considered the real-world representation of the entities [DMDH04]. If entities from two ontologies have highly similar instances, they are in turn considered as similar.

The techniques employed by current approaches work both on previously populated ontologies (i.e., ontologies which already come with defined instances) and on basis of instances which are collected and annotated with concepts from the ontology (e.g., sets of documents from the ontology domain). Once a sufficient set of instances is available, they can be incorporated in the matching process. We now briefly describe different approaches how instance information can be leveraged.

2.6.2.1 Element-Level Approaches

Basically, many approaches discussed in the context of ontology-based element-level matching, such as linguistic techniques in combination with language-based preprocessing, are also applied for instance-based matching. In addition, specific instance-based approaches have been devised.

**Linguistic approaches** use the previously described string-based similarity measures to compare instance values. Furthermore, IR techniques are often applied to infer frequent themes based on relative word frequencies observed in the instance data. Those themes (keywords) are then used to characterize ontology entities.

**Constraint-based approaches** use constraints in the form of patterns to compute the similarity between instances. For example, if two instance property values exhibit the same string pattern (e.g., a pattern used for display of dates such as “mm/dd/yyyy”), they are assigned a high similarity. This approach is computationally less expensive than the previous one, because a large part of possible constraints (patterns) can be disregarded at an early stage. An implementation of the described technique is presented, for example, in [EM07].

2.6.2.2 Probabilistic Approaches

So far, we have discussed only heuristic-based approaches to similarity computation. For example, similarities of entity names and instance values are computed at the element-level by applying a given string similarity measure, which computes the approximate similarity of two strings based on some heuristic. Considering instance-based matching, we sometimes observe probabilistic approaches and techniques adopted from machine learning, which constitute an alternative approach to the heuristic-based similarity computation techniques.

Instead of computing heuristic measures, those techniques consider the distribution of concepts over the instances, that is, they define the similarity between entities probabilistically. Often, different learners are employed to learn the distributions. The general approach is as follows (see, for example, [DMDH04]): On basis of the set of available instances, classifiers are trained for each concept. Then, for all concept pairs \( A, B \) from two ontologies, the joint probabilities \( P(A, B) \), \( P(\bar{A}, B) \), \( P(A, \bar{B}) \), and \( P(\bar{A}, \bar{B}) \) are computed, where \( P(A, B) \) is the probability that an instance of concept \( A \) also belongs to concept \( B \), etc. The computation of those joint probabilities is based on cross-classification of instances using the previously trained classifiers. Based on the calculated joint distributions, different similarity measures can be computed and used in the discovery of mappings.

The probabilistic approach follows closely the view that the semantics of ontology elements are modelled by their instances. Furthermore, it takes into account the uncertainty inherent in the problem of ontology matching. This technique is used, for example, in OMEN [MNJ04], GLUE [DMDH04], and the approach presented in [SE05].
2.6.3 Community-driven Ontology Matching

The community-driven approach to ontology matching is one that cannot be classified in any of the previously discussed groups. It is a complex approach relying both on an integration of several matching algorithms and on large communities of experts and users of ontologies to create and reuse ontology alignments. A community is viewed as a group of individuals with common interests, acting within common collaboration and communication environments. By incorporating user- and community-based information, the approach takes into account the high importance of direct human involvement in ontology management that has been argued in [ZKHF05].

The community-driven matching process is aimed towards an “evolution” of obtained mappings, facilitating contextualization and customization of alignments for specific communities and tasks. To this end, it embraces both conventional matching systems (thus leveraging the approaches described above), mappings directly provided by experts, and community-driven ontology management operations such as social network analysis. This way, information about the users is introduced in the process, such as their expertise, goals, and experiences with the ontologies, as well as social network information.

The ultimate goal of this approach is to obtain mappings that are tailored to user- and community-specific information. The idea behind this is that an alignment that is provided, affirmed, or induced by user input is likely to be valid for other users with similar user profile or application context. Moreover, it is likely to be more serviceable in that user’s community than a generic alignment would be. It is further argued in [ZS06] that alignments produced by such a community-driven approach will be more up-to-date and should more comprehensively cover the ontology domains than alignments produced by conventional techniques.

The advertised characteristics respond particularly well to the scalable and dynamic nature of the Semantic Web, thereby supporting its growth and evolvement. The authors of [ZS06] present a prototype that realizes the described approach in a collaborative environment. They state that experiments with the prototype yielded promising results.

2.7 Representative Tools

In the following, we present a small selection of matching tools to illustrate the diversity of the applied approaches and the integration of different approaches. Two of the presented tools — RiMOM and Falcon-AO — participated successfully in the OAEI 2006 alignment contest.

2.7.1 OLA

OLA (OWL Lite Alignment) [EV04] matches ontologies written in OWL Lite, leveraging the full spectrum of the provided language features. The system represents ontologies as graphs with different nodes and edges to represent ontology entities and language constructs. Using a generic integrative similarity function, OLA computes similarities between nodes of the same type (e.g., class nodes) as a weighted sum of their linked nodes in the graph (e.g., property nodes). Label similarities are computed based on simple string similarity measures and using auxiliary information from WordNet. To avoid cycles in the similarity computation, OLA uses a fix-point algorithm to iteratively propagate similarities computed from basic elements (e.g., entity labels) to adjacent nodes in the graph. The system includes a GUI enabling the user to configure the weights of the integrative similarity function and to display and edit obtained mappings. Furthermore, the system features a simple automatic weight configuration that distributes the weights in the integrative similarity function depending on the relative importance of different features in the ontology.
2.7.2 RiMOM

RiMOM [YLT06] applies an automatic two-step matching process integrating linguistic and machine learning approaches with graph-based similarity propagation. In the first phase of the process, RiMOM computes initial pairwise similarity values for all element pairs, applying basic string similarity measures. Furthermore, the system composes “documents” from a concept’s features such as its name, properties, and instance data. Using cross-classification, RiMOM then associates the documents of classes in the source ontology to those in the target ontology to compute probabilistic similarities (see Section 2.6.2). In the second phase, RiMOM applies a fix-point algorithm to propagate the initial similarities within class and property hierarchies, as well as between classes and properties.

The described matching process is preceded by a strategy selection in order to vary the impact of the two phases depending on the specific characteristics of the task. To that end, the system estimates the overall linguistic and structural similarity of two ontologies and applies the two phases depending on this estimate. For example, propagation within the class hierarchy is only applied in case of high structural similarity.

2.7.3 Falcon-AO

Falcon-AO [JHCQ05, HCZ⁺06] is a tool for automatic alignment of ontologies, which uses three individual matchers, considering both linguistic and structural information. The matcher I-Sub applies basic string similarity measures to compare label information. The V-Doc matcher assembles feature vectors from the words contained in entity descriptions and their neighbouring entities and computes their similarities in the vector space model [SM86]. Finally, the GMO matcher represents ontologies using RDF bipartite graphs and computes the structural similarities between those graphs [HJQW05].

A central controller component is responsible to integrate the three approaches to produce a final matching result. Using an approach similar to the one reported for RiMOM, the controller relies on the overall linguistic and structural comparability of the two ontologies to integrate the matchers. In particular, the influence of the linguistic-based V-Doc and I-Sub and the structure-based GMO on the final alignment is determined by the degree of linguistic comparability and structural comparability, respectively.

Another important feature of Falcon-AO is the partitioning component PBM. It facilitates matching of large ontologies by partitioning them into small blocks based on structural and linguistic similarity of the blocks. The individual blocks can then be matched efficiently.

2.7.4 An Information Retrieval Approach — iMapper

The iMapper tool [SG05] implements an approach that integrates instance-based matching with several complementary approaches. The matching process consists of two phases: Enrichment and mapping discovery. The enrichment phase exploits instance information to semantically enrich ontology elements. Instances are provided in the form of text documents, which are associated with the ontology elements using linguistic classifiers and user input. Based on the associated instances, feature vectors are build for each concept based on weighted word frequencies.

In the phase of mapping discovery, different approaches are used to compute pairwise similarities between the semantically enriched elements. Based on the cosine similarities between the element feature vectors, the system selects for each element in the source ontology the top k ranked elements in the target ontology. In a second step, the similarities of the selected element pairs are adjusted. For example, similarity values get a boost when the concept names of the elements are similar or when relations between them are discovered in WordNet. According to [SG05], iMapper performed well in the discovery and ranking of plausible mappings.
2.8 Summary

The evaluation we carried out in this chapter shows that there is an immense range of diverse approaches to ontology matching. Many of the approaches presented in Section 2.6 have been implemented in tools for (semi-)automatic ontology matching. The brief discussion of representative tools illustrated furthermore how diverse the application and integration of the different approaches can be realized.

Based on the obtained insights in the diversity of technologies, we identify the following issues as outstanding aspects of the current state of the art in ontology matching:

- **Leveraged information** Most systems using heuristic similarity computation rely largely on linguistic and structural features of the ontology entities. Some systems consider the specific semantics of ontologies, such as OLA, which leverages the complete OWL Lite language features. Others consider only certain aspects of ontologies, such as RiMOM, which uses relation semantics for similarity propagation between classes and their properties. It would be desirable to further investigate the utilization of ontology semantics in the matching process, as pointed out in [KS05].

- **Integration facilities and customization support** All state-of-the-art tools combine several approaches, leveraging diverse input information. We observe that the integration of applied techniques is done using both hybrid and composite matchers. Most systems apply a rather complex matching process. This process can be performed iteratively (e.g., OLA), combining approaches sequentially (e.g., RiMOM), executing and combining parallel matchers (e.g., Falcon-AO), or using some principal technique which is complemented by auxiliary matchers (e.g., iMapper). Some tools such as OLA allow a user to configure the matching process, for example, by providing weights. Others such as RiMOM apply a rigid process. The problem with such approaches is that they do not allow a user to account for different matching task characteristics, even when such knowledge is available. It is important to consider that not only the mismatches of the task at hand, but also the intended use of the alignment are important to determine a suitable matching approach [BSZ04]. This issue is not addressed in most of the literature. We hold that a flexible configuration of the matching process can help the user to introduce background knowledge about the task at hand.

- **Automation** In the last years, the automation of the ontology matching process has been increasingly addressed. Several of the discussed systems, such as OLA and RiMOM, apply some automatic configuration scheme trying to account for the characteristics of the matching task at hand. Configuration is usually based on some estimate of the overall similarity of ontologies with respect to some aspect. Even when no fully automatic approach is targeted, systems could leverage a semi-automatic configuration scheme. For example, a user could indicate the general task characteristics by specifying a set of criteria, based on which the matching system could then carry out the detailed configuration.

- **Utilization of auxiliary information** The utilization of lexica such as WordNet in the matching process seems a reasonable and promising approach. The idea encouraging this belief is that the naming of concepts is inevitably influenced by the natural linguistic context of human thinking. Thus, element names inhere additional, implicit semantics, which can be explored by using auxiliary background knowledge. There exist various approaches to the incorporation of lexica in the matching process, as reported, for example, in [KV04, BSZ04]. Another interesting idea is the integration of upper-level ontologies to provide a common semantic grounding. Resources such as the described alignment between WordNet and SUMO could be used as a natural language index to discover relations to common SUMO concepts. To our knowledge, as yet no system has realized this approach.

- **Characteristics of the discovered mappings** Most systems associate the discovered mappings with a confidence value, thus signalizing the supposed degree of similarity between the mapped elements and enabling a ranking of different mapping proposals. However, as pointed
out in [SdM06], hardly any system can produce precise semantic mappings. It would be desirable to further investigate how the discovery of exact semantic relations such as subconcept or containment relations can be facilitated. Such relations are likely to exist between entities of heterogeneous ontologies, since mismatches often occur in the scope of concepts and the granularity of modelling.

- **Probabilistic approaches to the matching problem** There exists a range of systems that define similarity between ontology entities probabilistically. This is generally achieved by considering the extension of ontologies, and associating instances with concepts from both ontologies based on cross-classification. The drawback of such approaches is that they rely on the existence of a sufficient set of instances. In order to adequately model the domain and its concepts, the instances should both cover the concept space and separate the different concepts well. Such instances must either be at hand in advance or they must be provided from external sources, either obtained manually [DMDH04] or automatically [PDYP05]. In cases where a sufficient pool of instances is available, the probabilistic approach seems promising, first, because it considers portions of reality to model the semantics of concepts and second, because it takes into account the uncertainty inherent in the matching task.

**Conclusions for this thesis** In the remainder of this thesis, we want to investigate some of the above-mentioned aspects further to incorporate them in our ontology matching methodology. First, we want to approach the discovery of mappings based on heuristic similarity computation. Although the probabilistic approach seems promising with respect to its interpretation of the matching problem, we will not address it in this work. As described above, applying such an approach implies that ontologies are populated with sufficient sets of instances, or such instances can be provided from appropriate sources, both of which we cannot assume.

To enable optimal matching results, we want to use several of the described approaches. In particular, we want to go beyond most current systems by exploiting the full semantics of OWL DL ontologies, thereby following the approach taken in [EV04]. Additionally, we aim to investigate whether the use of auxiliary information as provided by lexica can improve matching results. Furthermore, we highlight the importance of a flexible matching process configuration, which many current systems do not provide. We hold that this is one approach to allow a user to introduce context-dependent information in the matching process. This way, we can account better for the specific heterogeneities between ontologies and the problem of pragmatics in ontology matching.

Finally, although we identified the discovery of precise semantic mappings as an important aspect for current research, we will not investigate it in this thesis. We agree with the position defined in [SdM06], where it is argued that exact semantic relationships can only be inferred effectively based on a model-based approach to the matching problem.

In the following chapter, we will devise a dedicated ontology matching methodology addressing the stated objectives.
3 Towards a Flexible Ontology Matching Methodology

Concluding the presented overview of the state of the art in ontology matching, we identified a number of issues we view as decisive factors for a successful approach to the ontology matching problem. Based on those findings, we establish in this chapter a methodology for flexible matching of ontologies. After discussing the principle design goals we want to follow (Section 3.1), Section 3.2 outlines the general conceptual framework of our methodology. Then, we sequentially analyse the components of that framework, discussing the specific features of ontology semantics (Section 3.3) and addressing the realization of individual matching approaches that utilize those semantics (Section 3.4). Section 3.5 evaluates the integration of individual approaches into a complex matching process, and Section 3.6 presents functionality to increase the automation of the overall process. We summarize the chapter in Section 3.7.

3.1 Design Goals

This section discusses a number of principle design goals that shall set the course for the development of the ontology matching methodology.

1. **Matching between heterogeneous ontologies to improve interoperability:**
   The goal of ontology matching is to enable reconciliation of heterogeneous ontologies by finding mappings between their elements. The methodology developed in this work shall facilitate semi-automatic discovery of mappings. To achieve this goal optimally, we want to leverage existing work in the field of ontology matching, building on promising approaches as discussed in the previous chapter.

2. **Achieve a high-quality matching result:**
   The foremost goal of this work is the development of a methodology that produces high-quality ontology alignments. In particular, we want to achieve a solution that performs well in an objective and comprehensive evaluation. This presupposes that the devised methodology can be realized within the given time frame and can be evaluated properly. We want to evaluate the methodology against a benchmark series of ontology alignment tasks published by OAEI [OAE06]. The matching tasks in this test series address different types of mismatch, thus enabling us to analyse the influence of those mismatch types on the matching result, as well as on the performance of single matching approaches.

3. **Flexibility of the matching process**
   Our methodology shall enable a highly flexible matching process. To achieve this goal, it should integrate a large number of existing linguistic, structural and instance-based approaches to facilitate an optimal utilization of all available information. Indeed, all state-of-the-art tools for ontology matching approach the matching task by using combinations of different methods. However, most of the tools we investigated apply rather rigid combination strategies. In contrast to that, our methodology shall provide the user with various strategies to integrate the individual approaches. We hold that by enabling such a flexible integration, we can provide both for high customizability and good generic quality of the matching process.

   **High customizability:** The most apparent gain of a flexible ontology matching methodology is that it enables users to gear the matching process to task-specific requirements, because they can specify exactly how different approaches shall contribute to the overall result. Aspects which influence how the matching should be conducted include, for example, the task-specific mismatches...
and the intended use of the produced mappings. Based on such background knowledge, a user can induce which features are most likely to contribute to a good matching result and thus configure the process depending on the particular task. For example, if the linguistic overlap between two ontologies is high, the user might decide to employ a matching approach that is based on linguistic features and give it a high influence in the overall matching process.

**Generic applicability of the methodology**: A highly customizable methodology is at the same time also broadly applicable, in the sense that it can be tailored to fit individually a high number of specific application cases. On the other hand, and maybe less obviously, it also empowers the user to configure the process in such a way that it performs reasonably well for a wide variety of tasks, that is, to configure a generic matching process. For example, the configuration can be set up so that matchers that work well on different kinds of mismatches complement each other in the matching process, thus compensating individual weaknesses.

**Adaptation of the methodology to different time and memory requirements**: Due to their computational complexity, the various presented matching approaches impose different computational requirements, which influence the applicability of the respective approach. This resumes the considerations about different application contexts for ontology matching in Section 2.2. Although performance issues are outside the scope of this work, we observe that a flexible combination of approaches can help to adapt the process to nonfunctional restrictions. For example, a particular application scenario might only allow for very short computation periods, or the available memory might be constrained. In such a case, a lightweight matching process must be configured and employed. Conversely, if no such constraints are imposed the user might wish to exploit the features of the given ontologies as best as possible by using more complex measures, while accepting a potential drawback in efficiency.

4. **Exploiting semantic information and use of auxiliary sources**: Most matching approaches so far documented in the literature are based largely on linguistic and structural (hierarchical) information, which can be extracted from the entity labels and the hierarchies defined over the ontology concepts. Sometimes, instance-based techniques are used either as the main or a complementing approach. Many of the applied techniques resemble traditional approaches from the field of schema matching. While it is reasonable to apply schema matching approaches as part of an ontology matching methodology, they do not exploit the specific semantics that can be extracted from ontology constructs. Some current systems already leverage (part of the) ontological semantics, for example, RiMOM [YLT06] and OLA [EV04], which were described in Section 2.7.

While we want to make use of the aforementioned linguistic and structural approaches, the focus of our methodology shall be particularly on the utilization of the semantic information in ontology representations, more precisely, such information as can be expressed in OWL DL.

Furthermore, this work shall address the use of lexical auxiliary knowledge in combination with semantic background information in the form of (upper-level) reference ontologies. We expect that such information can improve matching results, particularly so in the context of ontologies with low syntactic and structural overlap.

**Ontology Language Support** In Section 2.4 we have illustrated the problem of language-level mismatches by comparing two current ontology representation languages and pointing to some mismatches between their features. Undoubtedly, enabling language interoperability through language normalization plays an important role in ontology reconciliation problems. It shall, however, not be the focus of this work.

At present, OWL is the de facto standard for ontology representation and has been adopted by many researchers as the basic interchange format. Furthermore, due to the collaborative process that has lead to the development of OWL, it can be regarded as a common denominator of the basic requirements for
ontology representation that arise in various application contexts. Most features of different languages can therefore be related in some way to those of OWL. Thus, considering the task of ontology matching in the context of knowledge sharing and interoperability, it makes sense to focus our attention to OWL as a commonly agreed-on standard.

Moreover, the focus of this work is on the development and evaluation of an ontology matching methodology. Therefore, we need a set of feasible test data, that is, (pairs of) ontologies to which we can apply our methods. Up to date, most ontologies that both constitute appropriate test cases and are publicly available — such as the ontologies in the OAEI benchmark — are written in OWL.

This said, the remainder of this chapter presents our concepts for a matching methodology that utilizes the language features of OWL DL [W3C04a].

### 3.2 Concept Outline

This section briefly sketches out the main aspects of our ontology matching methodology to give the reader a high-level insight. Figure 3.1 provides a conceptual overview of the components that make up the methodology. Below, we describe the different elements and relate how each of them corresponds with the design goals stated above.

We explain the methodology in a bottom-up approach, starting with a detailed discussion of the ontology descriptions we want to match (element description definition), and proceeding to considerations about how we can build simple and complex matching approaches on top of this basis (similarity definition and integration).

**Element Description Definition** In Section 1.4 we state that a mapping relates a pair of ontology elements that exhibit a strong semantic similarity. Furthermore, we have highlighted that our methodology shall particularly exploit the rich variety of semantic features used for concept descriptions in OWL DL. In the context of this work, we consider the semantics of an ontology element to be defined by the entirety of all features (information parts) in the ontology that contribute to the overall description of the element. Features can be direct constituents or related elements of the concerned element. For example, a named class has a name label as constituent, and its child classes and properties can be regarded as related elements. In Section 3.3 we elaborate in detail on the descriptions of different ontology elements, identifying the set of features we can use in the matching process, including auxiliary information from external sources. As illustrated in Figure 3.1, the definition of element descriptions is at the heart of the methodology, and all other components build on top of that definition.

**Similarity Definition** As stated above, we follow a similarity-based approach to mapping discovery, that is, elements from two ontologies will be related to each other based on their similarity. We can compute the similarity of elements based on different descriptive features (i.e., constituents and related elements). Section 3.4 discusses the definition of similarity measures based on the element descriptions. The described measures are realized within our methodology as a set of matchers. That is, a matcher can be regarded as a component that applies a particular similarity measure for the comparison of certain features of elements. For example, we can use a string-based matcher that computes similarities of element labels and comments, a matcher that compares class elements based on their children, and various other approaches discussed in Section 2.6. By including a broad range of matchers, we enable the user to apply different approaches to the matching problem. This increases the flexibility of the overall approach.

**Integration** As previously described, the matching methodology shall leverage different kinds of linguistic, hierarchic, and semantic information to enable a highly effective and precise discovery of mappings. We make this information available in the matching process as part of the element descriptions
Characteristics of the ontology matching task  

The problem of ontology matching is to a high degree influenced by requirements that arise from the types of mismatch that occur in a specific matching task. These requirements can be categorized into several types, such as:

1. **Ontological Mismatch**: This type of mismatch occurs when the ontologies being matched have different structures or concepts. The characteristics of this mismatch include:
   - **Heterogeneity**: The ontologies have different levels of granularity or detail.
   - **Semantic Shifts**: Certain concepts in one ontology may have different meanings or interpretations in another ontology.

2. **Conceptual Mismatch**: This type of mismatch occurs when the ontologies have overlapping concepts, but they are described differently. The characteristics of this mismatch include:
   - **Semantic Ambiguity**: Concepts with similar meanings are described using different terms.
   - **Synonymy**: Concepts with similar meanings are described using the same term but in different ontologies.

3. **Syntactic Mismatch**: This type of mismatch occurs when the ontologies have the same concepts but are described in different syntactic structures. The characteristics of this mismatch include:
   - **Vocabulary Discrepancy**: The ontologies use different terms to describe the same concept.
   - **Syntax Differences**: The ontologies use different syntactic structures to express the same concept.

To address these mismatches, ontology matching methodologies need to be flexible and capable of adapting to different scenarios. This involves:

- **Selection of Matchers**: Choosing the right matching techniques based on the type of mismatch.
- **Combination of Results**: Aggregating the results from different matchers to produce a comprehensive similarity score.
- **Configuration of System**: Setting up the system to handle the integration of different features in the matching process.

Thus, the matching process is subject to a relatively complex configuration, which is either conducted manually or automatically. Section 3.5 elaborates on those integration-related aspects of the methodology.
task, as well as the application context in which the matching is conducted. The former aspect determines which features can actually be leveraged in the discovery of mappings. For example, in case of a matching task where the concept hierarchies of the ontologies are completely different, the matching process should not use hierarchical information, because this could deteriorate the matching result. The application context, on the other hand, might impose restrictions on the kind of mappings to be discovered, depending, for example, on the intended use of the mappings. Thus, the characteristics of an ontology matching task influence considerably how the matching process should be configured, that is, which approaches should be integrated for a specific task.

3.3 Definition of Element Descriptions

We now elaborate on the definition of element descriptions, which forms the basis of our methodology, as shown in Figure 3.1. In our approach, which follows the view taken in [EV04], the semantics of an ontology element \( e \) is defined by the entire information in the ontology that contributes to the description of \( e \). We call such information the features of \( e \). Depending on the kind of ontology element considered, the set of possible features varies.

In this section, we first introduce the set of ontology elements we relate to throughout this chapter. Then, we discuss feature sets of different elements and show how those features can be viewed as contributions to the overall semantics of an element. Those considerations form the basis for an inclusion of different features in the matching process.

For a better understanding of the following sections, consider the ontology extract given in Listing 3.1:

Listing 3.1: Extract from an OWL ontology

This extract describes the three classes \textit{Employee}, \textit{Researcher}, and \textit{FacilityManager}, and two properties \textit{hasSalary} and \textit{hasColleague}, which both declare the class \textit{Employee} as their domain. For the data property \textit{hasSalary} we declare the XML Schema datatype \textit{positiveInteger} as range. Furthermore, a local class restriction specifies that for each instance of class \textit{Researcher}, the values of the symmetric property \textit{hasColleague} are restricted to class \texti{Researcher}. This expresses the (somewhat elitist) view that researchers only have other researchers as colleagues.

In the remainder of this chapter, we will refer back to this simple example to elucidate our concepts.
Ontology elements and entities  First, we want to introduce the basic ontology elements we consider in our methodology, relating them to the respective language features in OWL DL. For the sake of clarity, we distinguish between a) ontology elements and b) ontology entities. Within this thesis, the term element refers to all OWL DL ontology constructs for which we want to compute similarities in the matching process. We distinguish the following element types:

**Classes** comprise named classes as specified through the OWL construct owl:Class. In the above example, there are three named classes Employee, Researcher, and FacilityManager. We do not subsume anonymous classes under this element type.

**Properties** can be object properties and data properties, specified by the constructs owl:ObjectProperty and owl:DatatypeProperty, respectively. The former is used to describe relations between instances of two classes, the latter describes relations between instances of a class and RDF literals or XML Schema datatypes.

**Restrictions** are used to declare a constraint on a property for a particular class. Restrictions can be declared within the construct owl:Restriction, using owl:onProperty to indicate the restricted property. We can declare universal and existential range restrictions (owl:allValuesFrom, owl:someValuesFrom), cardinality restrictions (owl:minCardinality, owl:maxCardinality, owl:cardinality), and restrictions that specify classes based on particular property values (owl:hasValue). The example includes a universal range restriction restricting the values of property hasColleague to instances of class Researcher.

**Boolean combinations** of classes, so-called class expressions, can be built using the set operators owl:unionOf, owl:intersectionOf, and owl:complementOf. Operators can be nested, that is, complex class expressions can be built from other (anonymous) classes in turn. We do not consider this complex case in the course of this thesis.

The term entity shall refer in this thesis exclusively to named classes and properties. The example in Listing 3.1 contains three class and two property entities. In this work, we only consider mappings between the entities of an ontology.

3.3.1 Description Features of Ontology Elements

We now proceed elaborating on what we call the description of an element. In Figure 3.1 we show that an element description comprises the constituents and related elements of the concerned element.

- **Constituents** are information elements that form an integer part of an element definition and are not connected to other elements in the ontology, such as name or comment information.
- **Related elements** on the other hand are elements that are defined elsewhere in the ontology, such as the properties that are defined for a class, or the restrictions that are imposed on a property.

Table 3.1 summarizes for each type of considered ontology element the identified sets of constituents $C$ and related elements $R$, which combine to the set of description features $\mathcal{F} = C \cup R$.

In order to emphasize similar aspects of the features, they are assigned to feature groups that denote general aspects of ontology definitions, comprising, for example, all Hierarchy-related features. The considered feature groups are:

- **Label** denotes linguistic information intuitively describing an entity.
3.3. DEFINITION OF ELEMENT DESCRIPTIONS

- **Hierarchy** denotes information related to class and property hierarchies.
- **Relation** includes information about the properties of classes and the range and domain of properties, respectively.
- **Restriction** denotes information related to local class restrictions on properties.
- **Property** covers information about property characteristics, such as symmetry and transitivity.
- **Equivalence** denotes information about the equivalence of entities and disjointness of classes.
- **Boolean** denotes information about boolean combinations of classes.
- **Instance** considers class and property individual information.
- **Lexical context** denotes additional linguistic information retrieved from lexical auxiliary sources.
- **Semantic context** denotes additional semantic information related to an entity, for example, using (extended) upper ontologies.

**Example:** Consider the example of a class description given in Figure 3.2. Based on Listing 3.1, it shows the available features for class *Employee* in correspondence with the features listed in Table 3.1.

![Element Description Definition](image)

Figure 3.2: Description features of the class *Employee*

In addition to the listed features, we can also derive numeric data from the characteristics of most ontology elements, such as the number of subclasses, or the number of properties defined on a class. We will treat such statistical data as a constituent of the respective element. Since it is no explicit constituent, statistical data is not included in Table 3.1.

The remainder of this section discusses the different feature groups in detail. According to the classification in Section 2.5.2, we can distinguish them into internal and external information.

3.3.1.1 Internal Description Features

First, we consider such features that can be extracted directly from the ontology descriptions. We list the features in context of the different feature groups, describing their contribution to the semantics of elements. We exemplify the described features using the ontology extract given in Listing 3.1.
<table>
<thead>
<tr>
<th>Element type</th>
<th>Feature group</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Hierarchy</td>
<td>Child, parent, sibling classes</td>
</tr>
<tr>
<td></td>
<td>Relation</td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td>Restriction</td>
<td>Property restrictions applying to the class</td>
</tr>
<tr>
<td></td>
<td>Equivalence</td>
<td>Equivalent classes, disjoint classes</td>
</tr>
<tr>
<td></td>
<td>Instance</td>
<td>Instances of the class</td>
</tr>
<tr>
<td>Constituents</td>
<td>Label</td>
<td>Name and comment information</td>
</tr>
<tr>
<td></td>
<td>Lexical context</td>
<td>Synonyms, hypernyms, hyponyms in a lexical resource</td>
</tr>
<tr>
<td></td>
<td>Semantic context</td>
<td>Semantically related elements in an upper ontology</td>
</tr>
<tr>
<td>Property</td>
<td>Hierarchy</td>
<td>Child, parent, sibling properties</td>
</tr>
<tr>
<td></td>
<td>Relation</td>
<td>Domain classes, range classes (for object properties only)</td>
</tr>
<tr>
<td></td>
<td>Restriction</td>
<td>Restrictions defined on the property</td>
</tr>
<tr>
<td></td>
<td>Property</td>
<td>Inverse property</td>
</tr>
<tr>
<td></td>
<td>Equivalence</td>
<td>Equivalent properties</td>
</tr>
<tr>
<td></td>
<td>Instance</td>
<td>Instance values for the property</td>
</tr>
<tr>
<td>Constituents</td>
<td>Label</td>
<td>Name and comment information</td>
</tr>
<tr>
<td></td>
<td>Relation</td>
<td>Datatype (for data properties only)</td>
</tr>
<tr>
<td></td>
<td>Property</td>
<td>Functional, transitive, and symmetric property statements</td>
</tr>
<tr>
<td></td>
<td>Lexical context</td>
<td>Synonyms, hypernyms, hyponyms in a lexical resource</td>
</tr>
<tr>
<td></td>
<td>Semantic context</td>
<td>Semantically related elements in an upper ontology</td>
</tr>
<tr>
<td>Restriction</td>
<td>Restriction</td>
<td>Class to which the restriction applies, restricted property</td>
</tr>
<tr>
<td></td>
<td>Range class</td>
<td>Restricted property</td>
</tr>
<tr>
<td></td>
<td>Range datatype</td>
<td>Range class (for object property quantification only)</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean</td>
<td>Classes that are operands in the combination</td>
</tr>
<tr>
<td>Combination</td>
<td>Boolean</td>
<td>Operator type (union, intersection, or complement)</td>
</tr>
</tbody>
</table>

Table 3.1: Description features of ontology elements

**Label**  The most obvious information an element, in particular a class or property entity, provides, is naming and comment information. They represent part of the element semantics since they are in most cases “meaningful” labels given by humans, thus referring semantically to a certain linguistic context. Names and comments can be extracted from OWL ontologies through the constructs `rdf:ID`, `rdfs:label` and `rdfs:comment`.

**Example:** For the class `Employee`, we can extract the strings “Employee” and “A person employed by a company” as label and comment constituents.

**Hierarchy** The statements `rdfs:subClassOf` and `rdfs:subPropertyOf` are used in OWL to build up class and property generalization hierarchies, respectively. The premise of considering hierarchically related elements as part of an element description is that the semantics of a class or property are
(partially) determined also by the semantics of its neighbour (child, parent, or sibling) classes and properties in the ontology hierarchy. Derived statistics reflecting such hierarchical information of an element, such as the number of subclasses, can also be regarded as contributing to the element semantics.

**Example:** *Researcher* and *FacilityManager* are sibling classes of each other and child classes of *Employee*. Conversely, *Employee* is the parent class of *Researcher* and *FacilityManager*.

**Relation** Information about domains and ranges of properties, or datatypes in the case of datatype properties, are stated in OWL using the `rdfs:domain` and `rdfs:range` constructs. Obviously, such information contributes to the semantics of a property. Conversely, information about the properties defined on a class forms part of the semantics of this class.

**Example:** The property *hasColleague* has the domain *Employee*, property *hasSalary* has as range the XML Schema datatype `positiveInteger`.

**Restriction** Information that can be extracted from local class restrictions on properties forms an important part of the class description, insofar as it distinguishes a class from other classes that have similar properties but do not declare the same restrictions. We obtain such information through the constructs `owl:allValuesFrom`, `owl:someValuesFrom`, `owl:hasValue`, `owl:cardinality`, `owl:minCardinality`, and `owl:maxCardinality` in `owl:Restriction` elements. Conversely, restriction information apparently also contributes to the semantics of the restricted properties, distinguishing them semantically from similar properties that are not restricted.

**Example:** There is a restriction on the property *hasColleague* that applies to class *Researcher*. The restriction is a universal quantification restriction and restricts the property range to class *Researcher*.

**Property** In OWL DL we can specify a property as functional, symmetric or transitive by assigning the type `FunctionalProperty`, `SymmetricProperty`, or `TransitiveProperty`, respectively. Similar to the restriction features above, property characteristics contribute to the meaning of a property. Furthermore, using `owl:inverseOf`, a property can be related with its inverse property. We could consider such an inverse property either as a positive or negative contribution to the property semantics.

**Example:** The property *hasColleague* is declared as `SymmetricProperty`. Imagine we introduced another class *Manager* and a property *hasManager*. Obviously, this property would not be symmetric. Thus, its semantics with respect to the property characteristics would be distinct to that of *hasColleague*.

**Equivalence** Class and property equivalence is stated in OWL through the constructs `owl:equivalentClass` and `owl:equivalentProperty`. An individual belonging to some class also belongs to any class specified as equivalent to that first class. This functionality is used mainly to relate entities in different ontologies with each other when using them in a combined manner. However, it can also be applied to state equivalence of entities in the same ontology. From an equivalence statement we might infer that an entity shares the description of its respective equivalent entities.

In OWL DL we can also state disjointness between two classes using the construct `owl:disjointWith`. The meaning of this construct is that an individual belonging to a certain class

\[\text{The specification of a functional property corresponds to the information that the property has no more than one value (i.e., } maxCardinality = 1\). Thus, it can also be regarded as a constituent in the context of cardinality restrictions.\]
can not belong to any class that is declared disjoint to that first class. We identify two alternatives with respect to the way disjointness information can contribute to class semantics. First, we could simply regard the disjoint class as a related element without further semantics. Alternatively, we could take into account the actual semantics of the concept of disjointness, in the sense of stating an inequality relationship between two classes. In this case, we would regard a disjoint class as a negatively related element, as opposed to the positive semantics of the equivalence construct.

**Example:** Using the statement below, we can declare the class `Employee` equivalent to a class `Worker` from another ontology. Then, we could consider `Worker` a related element of `Employee` (with the special semantics of being an equivalent), and vice versa.

```xml
<owl:Class rdf:ID="Employee">
  <owl:equivalentClass rdf:resource="&otherOntology:Worker;"/>
</owl:Class>
```

**Boolean** We can specify boolean class combinations using the previously mentioned set operators. Classes that are operands in such a combination can be regarded as contributing to the semantics of the combination.

The `owl:complementOf` construct selects the part of an ontology extension that is not contained in the operand class. This suggests that the operand class could be regarded as a negatively related element of the combination, similar to the approach described for the `owl:disjointWith` construct. However, `owl:complementOf` is mostly used together with other (anonymous) classes within complex class expressions. Therefore, we doubt that we can evaluate it feasibly in the matching process.

**Example:** We could state that the class `Employee` includes both the extension (instances) of `Researcher` and `FacilityManager` using the `unionOf` construct. Then, we could regard the classes `Researcher` and `FacilityManager` as related (operand) elements of the union associated with `Employee`.

```xml
<owl:Class rdf:ID="Employee">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Researcher"/>
    <owl:Class rdf:about="#FacilityManager"/>
  </owl:unionOf>
</owl:Class>
```

**Instance** The set of instances that is declared for a given class is called the *extension* of that class. This term underlines that instances form an important part of the description of the class they belong to and of the properties they instantiate. Indeed, we can regard the extension of a class as a kind of “real world” representation of the class. This aspect has been acknowledged by many researchers, as discussed in Section 2.6.2.

**Example:** The class `FacilityManager` and property `hasSalary` can be instantiated as shown below. We can view the respective instance information (e.g., the value 1500 for the `hasSalary` property) as a contribution to the semantics of the two entities.

```xml
<FacilityManager rdf:ID="ACaretaker">
  <hasSalary rdf:datatype="&xsd:positiveInteger">1500</hasSalary>
</FacilityManager>
```
3.3. DEFINITION OF ELEMENT DESCRIPTIONS

Semantic context  Alternatively to using external semantic information from an upper ontology, we regard the information about the type of an element, such as Class and Property, as semantic information similar to the abstract concepts declared in an upper ontology, such as Entity or Relation. In this sense, we view the element type as an internal constituent of the concerned element.

Example: We can regard the type Class as a constituent of the entity Employee, and the type Property as constituent of hasColleague.

3.3.1.2 External Description Features

Apart from the information we can directly access from an ontology as an element’s descriptive features, this thesis investigates the exploitation of auxiliary lexical information in the matching process. We also discuss the possibility to incorporate semantic background knowledge, which is, for example, provided in the form of referenced upper ontologies. We refer to such auxiliary information as external description features of an ontology entity because they are not inherent in the ontologies. We describe this information as lexical context and semantic context, respectively. By lexical context we mean that additional linguistic information can be related to label information because terms reside in a similar linguistic context, for example, as neighbours in a taxonomy. Analogue, semantic context means that we can relate entities to concepts in an upper ontology, which convey additional semantic information.

Lexical context  For each entity in an ontology, label information can be extended by discovering related terms within auxiliary information sources, such as thesauri or lexica. As stated in Section 2.6, some researchers have already addressed the issue of lexical auxiliary information in dedicated matching approaches. Arguably, the most commonly used auxiliary source is WordNet [Mil93, CSL06]. We will therefore concentrate in the following considerations on the functionality realized in the WordNet database. WordNet distinguishes different senses of words and groups the words according to those senses in so-called synsets (sets of synonyms, that is, words that exhibit a similar sense), which are arranged in a taxonomy. Based on this organization, synonym (equal concept), hyponym (subconcept), and hypernym (superconcept) terms can be discovered for each term contained in the database. We consider such taxonomic lexical information as external constituents of an entity. This decision is motivated by the idea that the linguistic context has a high (subliminal) influence on the naming of entities in ontologies, thus forming an implicit part of entity descriptions.

Example: Querying WordNet, we can retrieve the terms “research worker” and “investigator” as synonyms of the term “Researcher”. Therefore, we could regard those two words as constituents of class Researcher.

Semantic context  In Section 2.6.1 we mentioned how the matching of ontologies can be facilitated when they extend a common high-level ontology. Unfortunately, the development of high-level ontologies has been realized only quite recently. Most ontologies have not been extended from such an ontology, and it is questionable whether researchers will commit to this approach in future development. As an alternative way to connect ontologies to upper ontologies, we mentioned the alignment of WordNet to the upper ontology SUMO [NP03]. The provided mappings can be utilized as a natural language index to the concepts of SUMO, enabling us to exploit the semantic background information in SUMO without the ontologies having to extend it. We could then regard related SUMO concepts as additional related elements, or include their label information as additional constituent of the concerned ontology element, respectively.

Due to the restricted time frame, we do not investigate this feature further within this thesis. Still, we
consider it an interesting approach to facilitate the matching process when no common reference ontology is given.

### 3.3.2 Representation of Ontology Element Descriptions – Ontology Graph Representation

In order to utilize the various features of ontology elements in the matching process, we need to make them available in an adequate representation. Such a representation must fulfill two major requirements: First, it must provide access to all ontology features we want to introduce in the similarity computation process. Second, it should have optimal memory requirements to allow for large ontologies to be loaded in the system.

Our methodology uses a special directed graph structure for representation of ontologies. A similar approach is taken, for example, in [EV04]. The authors describe the OL-Graph, a directed graph structure where different vertex and edge types represent the various constructs that can occur in OWL Lite ontologies. We use a simplified version of the concepts presented in [EV04].

Vertices in the graph are connected by means of directed edges, as shown in Figure 3.3. Additionally, each vertex and edge is assigned a particular type, according to the kind of ontology element and relation construct they represent.

![Figure 3.3: Typed directed edges connect typed vertices](image)

The different vertex and edge types in the graph allow access to the description features described above. Vertex types have self-explanatory names, which refer to the respective type of ontology element they represent, such as `ClassVertex`. Table 3.2 lists the edge types representing different relations between ontology elements. For each edge type, the respective sink and source vertex types in the graph are indicated.

Figure 3.4 illustrates how the descriptions in Listing 3.1 are represented as vertices and edges in compliance with the described graph structure.

<table>
<thead>
<tr>
<th>Edge type</th>
<th>Feature group</th>
<th>Vertex types</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IS_A</code></td>
<td>Hierarchy (children, parents)</td>
<td>Class, Class</td>
</tr>
<tr>
<td><code>IS_A</code></td>
<td>Hierarchy (children, parents)</td>
<td>Property, Property</td>
</tr>
<tr>
<td><code>PROPERTY</code></td>
<td>Relation (domain, property)</td>
<td>Class, Property</td>
</tr>
<tr>
<td><code>RANGE</code></td>
<td>Relation (property, range class)</td>
<td>Property, Class</td>
</tr>
<tr>
<td><code>INVERSE</code></td>
<td>Property (inverse, inverse)</td>
<td>Property, Property</td>
</tr>
<tr>
<td><code>RESTRICTS</code></td>
<td>Restriction (restriction, property)</td>
<td>Restriction, Property</td>
</tr>
<tr>
<td><code>APPLIES_TO</code></td>
<td>Restriction (restriction, class)</td>
<td>Restriction, Class</td>
</tr>
<tr>
<td><code>RESTRICTS_TO</code></td>
<td>Restriction (restriction, class)</td>
<td>Restriction, Class</td>
</tr>
<tr>
<td><code>EQUIVALENT</code></td>
<td>Equivalence (equivalent, equivalent)</td>
<td>Class, Class</td>
</tr>
<tr>
<td><code>EQUIVALENT</code></td>
<td>Equivalence (equivalent, equivalent)</td>
<td>Property, Property</td>
</tr>
<tr>
<td><code>DISJOINT</code></td>
<td>Disjointness (disjoint, disjoint)</td>
<td>Class, Class</td>
</tr>
<tr>
<td><code>BOOLEAN</code>²</td>
<td>Boolean (combination, operands)</td>
<td>Boolean, Class</td>
</tr>
</tbody>
</table>

Table 3.2: Vertex and edge types and their use in the ontology representation graph
3.4. Definition of Ontology Element Similarity

Since we aim to approach the matching task by considering similarities between pairs of ontology entities, we require a definition of the similarity of ontology elements. In this section, we build this definition on top of the element description definition discussed above. After introducing a general definition of description-based similarity, we discuss how the computation of similarities differs depending on whether element constituents or related elements are leveraged:

- Constituents can be compared based on the information they carry in the form of linguistic data (e.g., labels), numerical data (e.g., statistics), or constraint information (e.g., data and element types). Basic similarity computation is discussed in Section 3.4.2.
- Related elements can be compared based on their constituents or in turn by comparing their related.
elements. In this case, the computation of similarities amounts to a propagation of similarities between elements, which is discussed in Section 3.4.3.

Note that this differentiation corresponds to the distinction into element-level and structure-level matching approaches made in Section 2.6.1.

### 3.4.1 Description-based Similarity and Matchers

In the following we describe how the similarity of elements is defined on basis of their description features. This definition is based on the definition of element descriptions and the resulting sets of element description features \( F \) listed in Table 3.1. In its basic form, we define the similarity between two elements \( e_a \) and \( e_b \) with respect to a subset of selected features \( F_{selected} \subset F \) as

\[
sim(e_a, e_b) \equiv \sim(F_{selected}(e_a, e_b))
\]

where we write \( F_{selected}(e_a, e_b) \) to denote the pairings of selected descriptive features associated with the two elements, that is \( F_{selected}(e_a, e_b) = F_{selected}(e_a), F_{selected}(e_b) \), and \( \sim \) is a similarity function defined on the selected features. In effect, for each feature \( f \in F \), different similarity measures \( m \in M \) might be applied. \(^3\)

In the following, we denote the similarity of two elements \( e_a, e_b \) with respect to a specific feature \( f_i \in F_{selected} \) and a specific similarity measure \( m \), as \( \sim_{f_i,m}(e_a, e_b) \). Then, the similarity of two elements with respect to \( f_i \) and measure \( m \) is computed by applying \( m \) to the values extracted for feature \( f_i \) for elements \( e_a \) and \( e_b \):

\[
\sim_{f_i,m}(e_a, e_b) = \sim_m(f_i(e_a), f_i(e_b))
\]

**Example:** Consider as possible features that could be used to compute the similarity \( \sim(c_a, c_b) \) the labels, comments, properties, and parent classes of classes \( c_a \) and \( c_b \). Then, \( \sim(c_a, c_b) \) with respect to the label feature could be computed using the edit distance measure as \( \sim_{label,edit\_distance}(c_a, c_b) = \sim_{edit\_distance}(\text{label}(c_a, c_b)) \). Further similarities could be computed by applying a suitable measure \( m \) to the other feature values, that is, to the extracted comment data \( \sim_{comment,m} \), properties \( \sim_{property,m} \), and parent classes \( \sim_{parent,m} \).

**Combining constituent similarity values** Depending on the feature \( f_i \) we select for the computation of similarities, \( f_i(e_a) \) and \( f_i(e_b) \) can contain an arbitrary number of feature values each. For example, if \( f_i = \text{children} \), several child classes might be retrieved for two classes \( c_a, c_b \) each. Thus, to compute the similarity \( \sim_m(f_i(e_a), f_i(e_b)) \), the similarity measure \( m \) is first applied to compute the similarities between all pairs of extracted feature values \( f_i(e_a), f_i(e_b) \), resulting in \(|f_i(e_a)| \ast |f_i(e_b)| \) feature value similarities. In a second step, we must apply a combination operation in order to combine the feature value similarities into a final similarity value for the considered element pair \( e_a, e_b \). In the following, we assume that this combination is performed by taking the Average of the individual feature value similarities, unless stated otherwise.

\[
\sim_{f_i,m}(e_a, e_b) = \text{combination}(m(f_i(e_a), f_i(e_b)))
\]

\(^3\)Please note that different measures \( m \) can be applied on the same feature, and conversely, the same measure could be applied on different features. Hence, in the following definitions \( m \) denotes a freely chosen measure out of all available measures \( M \), provided that \( m \) is applicable on the respective feature.
3.4. DEFINITION OF ONTOLOGY ELEMENT SIMILARITY

Matchers  Based on the previous definitions, we formalize the concept of a matcher as the implementation of a specific $\text{sim}_{f_i, m}, f_i \in \mathcal{F}$. That is, a matcher is a component that applies a measure $m$ to compute similarities of its input elements based on their values for feature $f_i$. Note that we use the terms measure and matcher somewhat interchangeably in the following sections, depending on which captures more precisely the intended meaning in a given context (i.e., for referring to a method of similarity computation or a component within the methodology, respectively).

However, the concept of a matcher comprises more than just the similarity computation. In particular, a matcher component further utilizes the computed similarities to select a set of plausible correspondences between its input elements. The matcher can use different strategies for mapping selection, such as selecting the element pair with the highest similarities, a given number of highest-ranking pairs, or all pairs that exceed a given similarity threshold. Similar approaches are taken in the field of schema matching [Do05]. We do not want to discuss this aspect further at this point, since we are focused on the definition of ontology-specific approaches to similarity computation. We will address this issue briefly when relating to the implementation of the methodology.

3.4.2 Basic Similarity Measures

Table 3.3 provides a short overview and categorization of basic similarity measures, that is, such measures that work directly on the constituents of elements in the form of linguistic data, (semantic) type information, and numerical data. As mentioned above, different measures can be applied to compute the similarities of two feature values of a selected feature $f_i$, depending on how we define similarity in a given context. For example, for a particular matching task we might accept two entity labels as similar only when they actually are equal, whereas in another context substring equality might suffice.

The examples listed here reflect a range of measures applied by different matchers in our methodology. They could be extended easily with other algorithms.

<table>
<thead>
<tr>
<th>Example measures</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linguistic</strong></td>
<td></td>
</tr>
<tr>
<td>Heuristic</td>
<td>N-Gram, EditDistance, Substring, Prefix, Suffix Measures involving preprocessing (stemming, abbreviation expansion, etc.) Entity labels, comments, instance property values</td>
</tr>
<tr>
<td>Equality</td>
<td>Testing for equality In heuristic measures</td>
</tr>
<tr>
<td><strong>Lexical</strong></td>
<td>Degree of relatedness in a lexicon (e.g., WordNet) Entity labels</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td></td>
</tr>
<tr>
<td>User-provided</td>
<td>Compatibility tables Datatypes, ontology element types, restriction types</td>
</tr>
<tr>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Heuristic</td>
<td>Euclidean distance Depth of a class in the hierarchy, number of sub/superclasses, relations, cardinalities</td>
</tr>
<tr>
<td>Equality</td>
<td>Testing for equality In heuristic measures</td>
</tr>
<tr>
<td>User-provided</td>
<td>Numeric ranges Instance values</td>
</tr>
</tbody>
</table>

Table 3.3: Basic similarity measures
3. TOWARDS A FLEXIBLE ONTOLOGY MATCHING METHODOLOGY

3.4.3 Propagated Similarity

Computing similarities of two elements $e_1, e_2$ based on their related elements means that the similarities of the related elements are propagated to elements $e_1$ and $e_2$. Similarities of related elements can in turn be computed based on their related elements, and so forth. This amounts to a consecutive propagation of similarities between the different ontology elements.

Such an approach implies that a hierarchy of matchers is built, where a matcher $m_{n-1}$ is used by a matcher $m_n$ to compute similarities between the related elements of the input elements of $m_n$. At the lowest level, the computation is always based on some element constituent, that is, basic similarity measures are applied. This approach is applied to some degree by many current matching tools, either iteratively using a fix-point algorithm [YLT06], or applying single propagation steps [Do05]. Regarding the definition of element descriptions in Section 3.3 and the ontology graph structure presented in Section 3.3.2, we can view this technique in terms of propagating similarities along different edges of the graph.

Example: Consider a matcher that computes similarities of classes $c_a, c_b$ with respect to their properties, with the property similarities in turn being computed based on the restrictions that restrict them. Let the similarity of restrictions $r_a, r_b$ be computed based on a simple string comparison of the restriction types (e.g., “CardinalityRestriction” and “QuantificationRestriction”), that is,

$$sim_0 = sim_{\text{restr}_\text{type,string}_\text{equality}}(r_a, r_b) = sim_{\text{string}_\text{equality}}(\text{restr}_\text{type}(r_a, r_b)).$$

Then, the similarity of properties $p_a, p_b$ based on their restrictions is

$$sim_1 = sim_{\text{restrictions},sim_0}(p_a, p_b),$$

and the final similarity of classes $c_a, c_b$ is

$$sim_{\text{properties},sim_1}(c_a, c_b).$$

Observe how a hierarchy of matchers can be built based on sequential application of matchers as similarity measures of other matchers.

3.4.4 Similarity Specification for Ontology Elements

Above, we presented a definition of element similarity and described how basic and propagated similarity computation can be conducted on different features. Now we resume those considerations to show how the specific features identified for different element types in Section 3.3.1 can contribute to the similarity computation between those elements.

We consider the same groups of features as in Section 3.3. Mismatches that occur in a matching task are often related to one or several of the feature groups (e.g., Label-, or Hierarchy-related features), while other feature groups might not be affected (e.g., Relation features). Therefore, in the context of similarity computation, the features within a group can often be processed similarly.

Below, we discuss each of the feature groups in turn, resulting in a catalogue of similarity measures (matchers) $sim_{f_i,m}(e_a, e_b) = sim_m(f_i(e_a, e_b))$, where $e_a, e_b$ denote the type of element on which the matcher can be applied.

Note that we will not precisely specify the basic similarity measures used within the individual matchers, because their results depend strongly on the characteristics of the specific matching task, such as the degree of label mismatches, as pointed out before. Therefore, we leave the applied measure $m$ as a variation point, denoting only the kind of feature to which the measure must be applicable. Thus, we distinguish measures $m_{\text{str}}, m_{\text{num}}, m_{\text{comp}}, m_{\text{c}}, m_{\text{p}}, m_{\text{r}},$ and $m_{\text{b}}$. Measures $m_{\text{str}}, m_{\text{num}},$ and $m_{\text{comp}}$ refer to basic similarity measures applicable to strings, numeric data, or type information, as described in Section 3.4.2. The measures $m_{\text{c}}, m_{\text{p}}, m_{\text{r}},$ and $m_{\text{b}}$ are applied to compute similarities of classes, properties, restrictions, or boolean combinations, respectively.
3.4. Definition of Ontology Element Similarity

**Label**  Matchers can compare classes or properties based on their label or comment information by comparing the extracted strings. Correspondingly, similarity between classes \(c_a, c_b\) or properties \(p_a, p_b\) is computed as \(\text{sim}_{\text{label}, \text{m}, \text{e}}(e_a, e_b)\) or \(\text{sim}_{\text{comment}, \text{m}, \text{e}}(e_a, e_b)\), where \(e = c\) or \(e = p\), respectively.

**Hierarchy**  Hierarchical information of classes \(c_a, c_b\) can be used in the similarity computation process by propagating the similarity of classes to their respective child, parent, or sibling classes. Additionally, similarities can be propagated through several edges of the graph to all subsuming or subsumed classes. The same applies to properties \(p_a, p_b\). On this basis, hierarchy based similarities can be computed as \(\text{sim}_{\text{hierarchy}, \text{m}, \text{e}}(e_a, e_b)\), where \(e = c\) or \(e = p\), respectively, and \(\text{hierarchy} \in \{\text{children, parents, siblings, sub, super}\}\). Furthermore, statistic measures can be derived from the hierarchy, such as the number of child classes \#\text{children}. Then, similarities can be computed as \(\text{sim}_{\text{#hierarchy}, \text{m}, \text{num}}(e_a, e_b)\), where \(e = c\) or \(e = p\), respectively, and \(\text{#hierarchy} \in \{\#\text{children}, \#\text{parents}, \#\text{siblings}, \#\text{sub}, \#\text{super}\}\).

**Relation**  Information about the domains and ranges of properties \(p_a, p_b\) can be used to propagate the similarities of domain and range classes or datatypes (in case of data properties) to the properties. Conversely, for classes \(c_a, c_b\) similarities of their properties can be propagated to the classes. On this basis, relation-based similarity of properties \(p_a, p_b\) or classes \(c_a, c_b\) can be computed as \(\text{sim}_{\text{domain}, \text{m}, \text{e}}(p_a, p_b)\), \(\text{sim}_{\text{range}, \text{m}, \text{e}}(p_a, p_b)\), \(\text{sim}_{\text{datatype}, \text{m}, \text{comp}}(p_a, p_b)\), or \(\text{sim}_{\text{property}, \text{m}, \text{e}}(c_a, c_b)\), respectively. Again, we can further derive statistic measures from the number of domain or range classes of a property \(\#\text{domain}\), \(\#\text{range}\), or the number of properties a class has \(\#\text{property}\). The corresponding similarities are then defined as \(\text{sim}_{\#\text{domain}, \text{m}, \text{num}}(p_a, p_b)\), \(\text{sim}_{\#\text{range}, \text{m}, \text{num}}(p_a, p_b)\), or \(\text{sim}_{\#\text{property}, \text{m}, \text{num}}(c_a, c_b)\), respectively.

**Restriction**  Restriction information can be used to compute similarity of elements in various ways. First, we can compare restrictions \(r_a, r_b\) based on the type of the restriction, that is, whether it is a cardinality or a quantification restriction, and on its value, that is, to what cardinality or range it restricts a property. Furthermore, restrictions can be compared based on the classes and properties they apply to. On this basis, we define the following restriction similarities: \(\text{sim}_{\text{rest}, \text{m}, \text{type}, \text{m}, \text{comp}}(r_a, r_b)\), \(\text{sim}_{\text{card}, \text{m}, \text{num}}(r_a, r_b)\), \(\text{sim}_{\text{range}, \text{m}, \text{num}}(r_a, r_b)\), \(\text{sim}_{\text{property}, \text{m}, \text{p}}(r_a, r_b)\), and \(\text{sim}_{\text{class}, \text{m}, \text{e}}(r_a, r_b)\). Conversely, class and property similarities can be computed by comparing the restrictions that apply on them, or restrict them, respectively: \(\text{sim}_{\text{restricted}, \text{by}, \text{m}, \text{r}}(c_a, c_b)\), \(\text{sim}_{\text{restricted}, \text{by}, \text{m}, \text{p}}(p_a, p_b)\).

**Equivalence**  Information about equivalence can be used to contribute positively to similarities of class elements \(c_a, c_b\) or property elements \(p_a, p_b\) by propagating the similarities of equivalent entities to the compared entities. Usually, this propagation will reduce to one related element at most. On this basis, equivalence-based similarity is computed as \(\text{sim}_{\text{equivalent}, \text{m}, \text{e}}(e_a, e_b)\) where \(e = c\) or \(e = p\), respectively. For the similarity contribution of disjoint classes we have identified two alternatives with respect to the considerations in Section 3.3.1. First, disjoint classes can be considered as related elements similar to equivalent classes. In this case, similarities between classes \(c_a, c_b\) can be computed by propagating the similarities of their disjoint classes as \(\text{sim}_{\text{disjoint}, \text{m}, \text{e}}(c_a, c_b)\). Alternatively, we could consider the similarity contribution of disjoint classes as negative, taking into account the “negative” semantics of the disjointness construct. Information about disjointness would then be introduced to the computation of element similarities by negatively propagating the disjoint classes’ similarities with the compared class from the other ontology. Note that this implies a definition of similarity that is not compliant with our definition in Section 3.4.1. In particular, we would need to compute similarities by asymmetrically relating \(c_a\) with the disjoint classes of \(c_b\), and vice versa. Furthermore, to preserve the negative semantics, we would need to combine the obtained similarity
Property similarities can be computed based on whether the property has a certain characteristic such as being transitive, symmetric, or functional. We specify the respective similarities as $\text{sim}_{\text{char, char}}(p_a, p_b)$, where $\text{char} \in \{\text{functional, symmetric, transitive}\}$, and $m_{\text{char}}(\text{char}(p_a), \text{char}(p_b)) = 1$ if the concerned characteristic either does or does not apply to both $p_a$ and $p_b$, and 0 otherwise.

If a property specifies another property as its inverse, the inverse property can be regarded as a related element of the first property and similarities can be propagated as in the cases above. On this basis, we define the similarity $\text{sim}_{\text{inverse, prop}}(p_a, p_b)$. Note that this approach views the inverse property as a standard related element. Similar to the disjointness feature discussed above, we do not account for the “negative” semantics we might associate with the notion of an inverse.

Boolean

Boolean class combinations can be compared based on their operator, that is, whether the combination is a union, intersection, or complement, and by propagating similarities of the operand classes. Conversely, we can use combinations in the computation of class similarities by comparing the combinations they are operands to. Thus, we can define the following similarities:

- $\text{sim}_{\text{operator, comb}}(b_a, b_b)$
- $\text{sim}_{\text{operator, prop}}(b_a, b_b)$
- $\text{sim}_{\text{combination, prop}}(c_a, c_b)$

Lexical Context

Lexical information that can be extracted from the lexical database WordNet, such as synonym (equal concept), hypernym (superconcept) and hyponym (subconcept) terms of an entity label, can be used as a positive contribution to the similarity of entity labels $l_a, l_b$. Again, we can use this information in alternative ways. First, we define a complex similarity measure $m_{\text{taxrel, length(max)}}$, which performs several steps to compute similarities between labels $l_a, l_b$ based on their relatedness in the WordNet hierarchy and a threshold $\text{max}$. Having retrieved for both labels the corresponding WordNet synsets $\text{synset}_a = \text{synset}(l_a)$ and $\text{synset}_b = \text{synset}(l_b)$, we search for relations between those synsets in the WordNet taxonomy, that is, we try to find a path of hypernym/hyponym relations between $\text{synset}_a$ and $\text{synset}_b$. Then, the measure $m_{\text{taxrel, length(max)}}$ determines the similarity of $l_a$ and $l_b$ based on the length of the shortest discovered relation path relative to the specified maximum path length $\text{max}$. In case two words are associated with the same synset, they are regarded as synonym and the corresponding relation depth is 0; if one sense is hyponym to another, the depth is 1, etc.

According to this definition, similarity between two entities $e_a, e_b$ is defined as $\text{sim}_{\text{label, taxrel, length(max)}}(e_a, e_b)$.5

Example: Consider two labels $l_a, l_b$. Label $l_a$ is found to be a hypernym of label $l_b$ in WordNet. The path length between the two labels in the taxonomy is thus 1. Consider two values $\text{max} = 1$ and $\text{max}_0 = 0$ for the maximum path length. If measure $m_{\text{taxrel, length(max)}}$ is applied, the computed similarity value might, for example, be 0.5. If $m_{\text{taxrel, length(max)}}$ is applied, only synonym relationships are considered as sufficient for similarity, and therefore the similarity between $l_a$ and $l_b$ amounts to 0.

---

4 As stated above, we focus our considerations on WordNet, and ignore other possible sources such as domain specific thesauri or abbreviation lists, which could be leveraged similarly.

5 Note that the described approach constitutes a basic similarity measure, insofar as we apply it directly on the label elements. However, the measure internally computes similarities based on the synsets associated with the label terms, which are actually related elements. Thus, we could also describe the approach in terms of similarity propagation.
Alternatively to the described approach, lexical resources that organize their entries in a taxonomy could also be utilized for the discovery of superclass, subclass or sibling relations between two ontology elements. If \( \text{synset}_a = \text{synset}(l_a) \) is found to be in a hypernym, hyponym, or sibling relation to \( \text{synset}_b = \text{synset}(l_b) \), we might induce from this observation that there exists a corresponding relation between the classes associated with the labels \( l_a \) and \( l_b \). At present, we do not consider specific semantic mappings such as superclass and subclass correspondences, thus we do not include this approach in our methodology.

**Semantic Context — Similarity of element types** In Section 3.3.1 we mentioned that we regard the type of ontology elements as semantic information similar to the abstract classes defined in upper ontologies.

The way we process ontology elements of different types determines to a large extend the result of a matching process. More precisely, we have to decide to which degree we want to allow mappings between elements of different types. If we ignore information about the type of ontology elements, all possible combinations of ontology elements are considered potential mappings. For example, classes might be mapped to properties if the applied matcher computes a high enough similarity. On the other hand, we can aim at a separation between element types, that is, we can enforce that only elements of the same or similar type are considered potential mapping pairs. This is a reasonable approach in most cases and reduces the complexity of the matching task considerably, since less similarities have to be computed.

Both approaches can be realized applying a similarity measure that compares the types of ontology elements based on information about their compatibility, which is provided a priori. That is, the element type similarity of two elements \( e_a, e_b \) can be computed as \( \text{sim}_{\text{ele},\text{Type,m}}(e_a, e_b) \). Using such an approach, we can define strict element type separation by assigning a similarity of 0 if the types are different. However, sometimes a less strict approach might be desirable, thereby, for example, allowing mappings between object properties and data properties, or even between classes and object properties. The degree to which we want to enforce separation of types can differ, depending on the modelling paradigms used in the ontologies at hand. An address, for example, could be modelled as an object property in one ontology and as a simple string, that is, a data property, in another.

For the proposed methodology, we chose to apply by default a partial separation of ontology types, where classes are considered separate from properties, but object and data properties are considered potential mapping pairs. That is, we apply the element type similarity measure as a pre-selective matcher in the matching process.

### 3.5 Integration of Individual Approaches

In the previous section, we introduced different matchers that compute similarities based on various available description features. This allows us to utilize a broad range of semantic information, which we have stated as a central design goal at the beginning of this chapter. Also, it provides for a certain flexibility of the methodology, since we can apply different similarity measures on a given feature.

Often, it can be favourable to integrate different matchers in a complex matching process, so as to leverage several features of an element in parallel. This relates to the concept of a combined matcher, as discussed in Section 2.6. The combination of matchers can be conducted in various ways. This chapter discusses the integration functionality our methodology provides.

To produce high quality alignments for a given matching task, it is not sufficient to consider all the features that could provide useful information in the matching process. As discussed in Section 2.3, matching tasks can exhibit a broad range of ontology mismatches. Thus, observing a wide range of diverse ontology matching problems, we must also consider in what kind of problems the identified features actually do benefit the discovery of mappings, and choose the matchers accordingly. Once we
have determined the matchers we want to apply, we must delve into the details of how exactly they shall be integrated into a coherent matching process to account optimally for the characteristics of a specific matching task.

In the remainder of this section, we describe how our matching methodology allows flexible combination of individual matching approaches so that the user can customize the matching process to task-specific needs.

**Combination of Matching Approaches** In Section 3.1 we argued that we strive for high flexibility in the matching process, since this enables both customization and generic applicability of the matching methodology. Following this objective, we must ensure that the choice of individual matchers to be combined and their integration in a matching process is highly flexible and user-configurable.

As mentioned in Section 2.6, we can distinguish two approaches for the combination of matchers [RB01]: The *hybrid matcher* and the *composite matcher* approach.

**Hybrid matchers** perform the computation of similarity values between two elements by directly combining different approaches working on different features of the elements. They return a single similarity result from which the mappings are then extracted. The hybrid matcher approach can offer better performance than the composite approach because multiple criteria can be evaluated on each ontology element in one pass, and the results can be joined in a complex way, yielding higher-quality matching results.

**Composite matchers** on the other hand combine the results of different independent individual matchers, which can be hybrid and composite matchers in turn. This concept is described in detail in [Do05]. The composite matching approach is much more flexible than the hybrid approach, enabling the user to combine matchers at will. This comes at the price of a potentially less refined combination of the single approaches.

In our methodology, we apply the composite matcher approach, which corresponds well with the modular definition of element similarity we presented in the previous section. Moreover, the composite approach enables the targeted flexibility of matching process customization.

**3.5.1 Composite Matcher Approach**

Figure 3.5 illustrates how the matching process is conducted when a composite matcher is applied.

In the configuration of a composite matcher $m_{\text{composite}}$, the user can select a set of $k$ matchers as component matchers of the composite matcher. When executed, those component matchers independently compute similarities $sim_j = sim_{f_i,m_j, j = (0, \ldots k)}$ based on a selected feature $f_i \in F$, and an applicable measure $m$ each. As discussed in Section 3.4.1, $m$ can be either a basic similarity measure, or a propagation-based measure (matcher), in turn.

In order to produce a single similarity result, which can finally be used to extract mappings, the set of $k$ matcher specific similarity results must be *aggregated* for each pair of elements $e_a, e_b$:

$$sim(e_a, e_b) = \text{aggregation}(sim_0(e_a, e_b), \ldots, sim_k(e_a, e_b))$$

The applied *aggregation* operator determines how the final matching is computed. It can take various forms, thus enabling a user to specify flexibly how individual matcher results shall influence the overall similarity result.

In the following, we discuss how our methodology provides for a flexible integration of individual matching approaches.
3.5. INTEGRATION OF INDIVIDUAL APPROACHES

3.5.2 Basic Aggregation Operators

When a matcher is configured as a composite of a set $M$ of individual component matchers, as described above, the component results must be aggregated into a single similarity value. To this end, different aggregation operators can be used. Simple approaches include taking the maximum, minimum, or a weighted sum of component similarity values, as applied, for example, in [Do05].

**Maximum** returns the maximum similarity value for elements $e_a$ and $e_b$ from the results of all constituent matchers. It can be used when matchers shall complement each other optimistically, that is, when the user considers it sufficient that one matcher yields a high similarity value.

$$Max(e_a, e_b) = \max_{m \in M} sim(e_a, e_b, m)$$

**Minimum** returns the lowest similarity value of elements $e_a$ and $e_b$ from the results of all constituent matchers, that is, a pessimistic result. It can be applied when we consider it necessary that several similarity constraints are met, that is, that all constituent matchers yield a high similarity.

$$Min(e_a, e_b) = \min_{m \in M} sim(e_a, e_b, m)$$

**Weighted** returns a weighted sum of all similarity values, that is, all matchers influence the match result according to their expected importance, which is represented by the weights. Taking the Average of all results is a special case of **Weighted**, where all weights are equal, that is, $w_m = \frac{1}{M}$.

$$Weighted(e_a, e_b) = \sum_{m \in M} w_m * sim(e_a, e_b, m), \sum_{m \in M} w_m = 1$$

Using those operators, we can exploit and complement flexibly the benefits of matchers that leverage different features of ontology elements.

![Figure 3.5: The composite matching process](image)
Further operators can be considered. In Section 3.4 we described how some element features, such as the inverse of a property, exhibit semantics that can be interpreted as negative contributions to the similarity of two elements. To allow for an integration of negative similarity contributions, we apply the concept of distance. That is, instead of the similarity, we consider the dissimilarity of elements. Given an interval \([0 \ldots 1]\) for both similarity and distance values, a similarity value \(sim(e_a, e_b)\) can easily be converted into the corresponding distance. We compute \(dist(e_a, e_b) = 1 - sim(e_a, e_b)\), so that high similarities contribute small values to the final result, and vice versa. Analogue to the similarity-based operators above, we can apply the operators \(MaxDistance\), \(MinDistance\), and \(WeightedDistance\).

### 3.5.3 A Hybrid Flavour: Dependency of Matchers

In general, the integration of matching approaches is done very loosely in the composite approach, which brings the benefit of high flexibility. In contrast to that, hybrid matchers directly integrate different approaches, and therefore enable a more advanced computation of similarity based on several matching criteria. In [RB01] the authors state that hybrid matchers “should provide better match candidates plus better performance than the separate execution of multiple matchers. Effectiveness may be improved because poor match candidates matching only one of several criteria can be filtered out early”.

This said, it seems promising to investigate into an extension of the basic composite integration functionality by allowing some kind of dependency between constituent matchers. By declaring a matcher dependent on another matcher, we mean that the results of the second matcher shall only contribute to the overall similarity computation if the results of the first matcher fulfil a given criteria, such as exceeding or falling below a specified threshold value. We suggest that this approach introduces a “hybrid” facet to the matching process and extends the flexibility of integration. By making matchers depend on other matchers, we can introduce precedence constraints of certain criteria over others. Figure 3.6 illustrates the concept of a dependent matcher, for which we define:

- Matchers \(matcher_{base}\) and \(matcher_{dep}\) are constituent matchers of a composite matcher \(m\). The similarity result of \(matcher_{base}\) and \(matcher_{dep}\) for elements \(e_a, e_b\) is represented by \(sim_{base}(e_a, e_b)\) and \(sim_{dep}(e_a, e_b)\), respectively.
- Matcher \(m\) is further configured with a constraint \(constraint_{base}\), which specifies that \(sim_{base}(e_a, e_b)\) must either exceed or stay below a given \(threshold_{base}\).
- In the matching process, the similarity result \(sim_{dep}(e_a, e_b)\) is only included in the final similarity result if \(sim_{base}(e_a, e_b)\) satisfies \(constraint_{base}\). If the given constraint is not met, the final result is determined solely by \(sim_{base}(e_a, e_b)\).

**Example:** Consider a matching task where the user knows that both ontologies have a considerable linguistic overlap, that is, many elements of the ontologies are named similar. However, some concepts in the target ontology are named different, for example, using synonyms or hypernyms of the words in the source ontology. Based on that information, a user might now decide to use a combination of a string-based matcher (e.g., EditDistance) and a lexical matcher exploiting linguistic relations in an auxiliary lexicon. However, such a lexical matcher can also return results that are unfeasible in the given context. If the user is aware of that drawback, he can decide to give precedence to the string-based matcher in order to first determine string similarities and then apply the lexical matcher only on such element pairs where the string similarities fall below a predefined similarity value. Then, the results of the lexical matcher can not corrupt previously obtained string-based similarity results for a given element pair, provided they are high enough.
3.6 Enabling Generic Matching through Automatic Matcher Configuration

The integration functionality described in the previous section — the use of aggregation operators to combine results of individual matchers and the definition of dependency between matchers — are effective means allowing a user to tailor the matching process to a specific matching task. Apart from the high customizability, we highlighted the generic applicability of a matching process as another requirement that motivates a flexible integration approach. Manual matcher configuration fulfills this requirement in scenarios where a user can determine a configuration that suits the matching task. However, we cannot make use of the provided configuration facilities in cases where an automatic matching process is required, that is, where no manual configuration can be conducted before the match process is executed. In such cases, an optimal generic configuration is required.

Therefore, this work also investigates how the matching process can be configured automatically by evaluating the characteristics of the concerned matching task and configuring the process accordingly. Since the proposed methodology provides very flexible configuration functionality, there are different aspects we could consider in the context of automatic configuration:

Selection of matchers: To determine which features shall be integrated in the matching process, an automatic configuration can select a particular set of constituent matchers. For example, if the matching task exhibits strong structural heterogeneity, the configuration could exclude structure-based matchers.

Specification of weights: Different weights can be assigned to the applied constituent matchers to vary the influence of different features depending on how valuable they are considered for the discovery of mappings. For example, we might wish to assign higher weights to label- than to structure-based matchers.

Parameterization of basic similarity measures: By varying parameters of similarity measures, we can influence how strictly a given similarity measure is applied within a matcher. For example, we could configure the measure N-Gram with different values N.

In the context of this work, we investigate only the first configuration aspect, that is, the selection of constituent matchers to be integrated in the matching process. The remaining issues involve an even...
higher degree of freedom in the configuration. Therefore, it becomes a very hard and expensive task to elaborate feasible configuration rules for different task characteristics. Such a complex issue cannot be approached within this work due to restrictions in time and suitable test resources, that is, data that would be required to train and test the configuration approaches eventually devised.

Generally speaking, the automatic configuration facility of our methodology is designed as a simple decision mechanism, which selects different matcher components for integration in a complex matching process based on the general characteristics of the matching task. In the approach presented here, those characteristics are reflected by estimating the overall similarities of the labels \( \text{sim}_{\text{label}} \) and structure \( \text{sim}_{\text{struct}} \) of two ontologies. A similar approach was taken in [YLT06] (see Section 2.7.2). We adopt the idea to our flexible configuration scheme.

To this end, we devise a dedicated matcher, which performs matcher selection based on a given configuration and the aforementioned estimated ontology similarities. We discuss this approach in the next section. Therein, we also address an alternative approach to apply the configuration in a global manner. After that, we show how the task characteristics in the form of the estimated label and structure similarity are obtained.

### 3.6.1 StrategyMatcher Configuration

To realize automatic matcher configuration, we devise a component called StrategyMatcher, which is configured to apply a particular selection of matchers depending on the characteristics of a given task. The matcher is illustrated in Figure 3.7. As stated above, the configuration defines simple decision rules to determine how the matching process shall be performed. In particular, we specify three sets of matchers: \( M_{\text{default}} \), \( M_{\text{label}} \), and \( M_{\text{structure}} \). Whereas \( M_{\text{default}} \) contains matchers we consider to be generally applicable, \( M_{\text{label}} \) and \( M_{\text{structure}} \) contain such matchers we only want to apply on ontologies which are highly similar in their linguistic or structural features, respectively. To this end, we furthermore specify thresholds \( t_{\text{label}} \) and \( t_{\text{struct}} \). Based on those thresholds, we determine which matchers are executed by the StrategyMatcher, given a specific matching task. Whereas \( M_{\text{default}} \) is applied by default, the set of label and structure matchers is only included in the matching process if the computed similarity values \( \text{sim}_{\text{label}} \) and \( \text{sim}_{\text{struct}} \) exceed the configured thresholds \( t_{\text{label}} \) and \( t_{\text{struct}} \), respectively.

Clearly, even with the described functionality for automatic matcher configuration, we still need human intervention in order to determine the rules that shall be applied in the configuration. The user must determine feasible sets of label- and structure-based matchers, as well as adequate thresholds, so that the resulting matcher configuration performs well. This is a nontrivial task, since the user must possibly take into account the constituent matchers of the chosen matchers in turn.

**Example:** A StrategyMatcher could be configured with the following matcher sets and thresholds:

- \( M_{\text{default}} = \{ \text{sim}\_\text{property}, \text{m}_\text{datatype}, \text{sim}\_\text{restriction}, \text{m}_\text{rest}\_\text{type} \} \)
- \( M_{\text{label}} = \{ \text{sim}\_\text{self}, \text{m}_\text{label}, \text{sim}\_\text{property}, \text{m}_\text{label} \} \)
- \( M_{\text{structure}} = \{ \text{sim}\_\text{property}, \text{m}_p, \text{sim}\_\#\text{children}, \text{m}_\text{num} \} \)
- \( t_{\text{label}} = 0.2, t_{\text{struct}} = 0.3 \).

Using this StrategyMatcher, a matching process would include as constituent matchers by default the set \( M_{\text{default}} \). If the value obtained for \( \text{sim}_{\text{label}} \) exceeds 0.2, the set \( M_{\text{label}} \) will also be included, so that similarities would be computed based on the labels of the respective entities and of their properties. If the value for \( \text{sim}_{\text{struct}} \) exceeds 0.3, the set \( M_{\text{structure}} \) will be included. The matcher \( \text{sim}\_\#\text{children}, \text{m}_\text{num} \) computes similarities based on purely structural information (i.e., the number of children of an element). However, this is not definite for \( \text{sim}\_\text{property}, \text{m}_p \): If the measure \( m_p \) uses label information to compute property similarities, this would introduce a linguistic approach in the set \( M_{\text{structure}} \).
Global automatic configuration of the matching process Using the described approach, the user can configure different StrategyMatcher matchers, and then select one of them for the matching process. When the matcher is executed, the configuration settings are used for automatic matcher selection based on the current task characteristics. Alternatively, we can specify global configuration rules that apply to the overall matching process.

In the global approach, the configuration procedure is also based on the estimation of task-specific label and structure similarities. However, the user does not need to configure and use any specific matcher, but just applies normal composite matchers. The information about which matchers are label-based and structure-based is defined a priori for every available matcher. When the user executes a composite matcher, the system automatically excludes all such matchers that do not comply with the estimated task characteristics. To this end, task similarities $sim_{label}$ and $sim_{struct}$ are determined and compared against (globally configured) threshold values $t_{label}$ and $t_{struct}$. If the thresholds are not reached, the matchers that are marked as label-based or structure-based are excluded from the matching process, respectively. For example, if for a given task $sim_{struct} < t_{struct}$, then all matchers marked as structure-based will be excluded from the process, and thus the combined results of the executed composite matcher are computed without their similarity contribution. Using this approach, the user does not need to delve into the details of the applied matchers, since the “unsuitable” constituent matchers are automatically excluded from the matching process.

**Example:** Consider the following global configuration settings, where $GlobalConfiguration$ includes all available matchers, marking them as label- or structure-based:

$$GlobalConfiguration(matcher, structurebased, labelbased) = \{(m_{label}, false, true), (m_{#rel}, false, false), (m_{#children}, true, false), (m_{children}, true, true)\}.$$
**Towards a Flexible Ontology Matching Methodology**

$t_{\text{label}} = 0.25, t_{\text{struct}} = 0.25$.

Now, consider a matching task where $\text{sim}_{\text{label}} = 0.5$ and $\text{sim}_{\text{struct}} = 0.1$, that is, $t_{\text{label}}$ is exceeded whereas $t_{\text{struct}}$ is not. The user applies a composite matcher that applies as constituent matchers $m_{\text{label}}, m_{\#\text{rel}}, m_{\#\text{children}}$, and $m_{\text{children}}$. Now, in a normal matching process, all matchers would be executed and the results aggregated into the final similarity result. With the global automatic configuration activated, only $m_{\text{label}}$ will be executed, because for all other matchers the configuration rule is not satisfied: They are marked as structure-based matchers, but the computed $\text{sim}_{\text{struct}}$ is too low, and thus structure-based matchers are not applicable.

### 3.6.2 Determining Task Characteristics

Before the configuration mechanism can select suitable matchers for a given matching task, the characteristics of the matching task must first be determined. This information could be provided by a user or computed based on selected information from the ontologies.

**User-provided task characteristics** A user can provide information about the nature of the task insofar as such information is known to him. Based on the user’s estimation of the general task characteristics, the system could then apply the automatic configuration, as described above. Although the user would thus be involved in the configuration process, there would still be a certain degree of automation in that the respective matchers are selected automatically. In effect, such an approach would “delegate” the details of the configuration to the system, since the user only needs to give information about the task characteristics instead of conducting a detailed configuration.

For example, consider a task where the ontologies have a very low linguistic overlap, that is, their concept and property names are very dissimilar and thus do not contribute useful information to the matching process. Based on such knowledge, a user can inform the system that the value for $\text{sim}_{\text{label}}$ is very low. Then, the system could configure the matching process by excluding label-based matchers.

**Heuristic computation of ontology similarities** The previous scenario only applies when a user can interact with the system and has enough information about the nature of the ontologies. Considering that we want to enable a maximum of automation in the matching process, we need to determine the task characteristics automatically. This can be done by an approximate comparison of the two ontologies with respect to the features we want to evaluate in the configuration process.

Tang et. al. [YLT06] describe a similar approach, which they use in their tool RiMOM. As described in Section 2.7.2, RiMOM selects matching strategies based on the general characteristics of two ontologies. The important difference to our methodology is that RiMOM configures the matching process so as to determine whether a certain propagation strategy shall be applied or not, but the applied strategy cannot be assembled flexibly.

In the following, we describe how the approximate structural and label similarities of two ontologies $O_A$ and $O_B$ are estimated using the measures proposed in [YLT06].

The label similarity factor $\text{sim}_{\text{label}}$ is defined as

$$\text{sim}_{\text{label}} = \frac{\#\text{same label}}{\max(\#c_a, \#c_b)}$$

where $\#c_a$ and $\#c_b$ reflect the number of classes in $O_A$ and $O_B$, respectively, and $\#\text{same label}$ reflects the number of classes from $O_A$ and $O_B$ that have the same — or, depending on the applied similarity measure, highly similar — labels. Naturally, the measure used in this estimation step should be very simply so as to incur only low additional costs to the overall matching process, such as testing for equality of label strings or reasonably large substrings.
Structure similarity is defined as

\[
sim_{\text{struct}} = \frac{\# \text{common concept}}{\max(\#\text{nonleaf } O_A, \#\text{nonleaf } O_B)}
\]

where \( \#\text{nonleaf } O_A \) and \( \#\text{nonleaf } O_B \) reflect the number of classes that have children in \( O_A \) and \( O_B \), respectively, and \( \#\text{common concept} \) reflects the number of classes that are highly similar with respect to their structural statistics, such as their depth in the concept hierarchy, or their number of children.

Based on the resulting values \( \sim_{\text{label}} \) and \( \sim_{\text{struct}} \) and the previously specified selection rules in the StrategyMatcher configuration or the global configuration, the configuration mechanism determines the set of matchers that contribute to the overall matching result.

Of course, the aim of automatic matcher configuration is that the matching process is conducted in such a way that it corresponds well with the task characteristics. However, it is apparent that the estimated similarities \( \sim_{\text{label}} \) and \( \sim_{\text{struct}} \) can only very roughly reflect the true degree of congruence between two ontologies. It is easy to imagine tasks where \( \sim_{\text{label}} \) or \( \sim_{\text{struct}} \) yield a high value, although the true congruence of the ontologies with respect to the labels or structure is very low. In the worst case, this could then result in an unreasonable configuration of the matching process.

### 3.7 Summary

This chapter presented the concepts for a flexible ontology matching methodology that integrates a large variety of diverse matching approaches. In particular, we discussed how our methodology can leverage the broad range of semantics expressed in OWL DL ontologies. We described how the ontology elements that are considered in the process of mapping discovery are effectively defined by their set of descriptive features and how similarity measures can be defined over the element descriptions. The discussed similarity measures are realized within the methodology as individual matchers. We facilitate the combination of such matchers into complex composite matchers, which integrate the results of the individual matchers. We presented various strategies that can be used for this integration, ranging from simple aggregation operators to automatic configuration of the matching process.

Concluding, the presented methodology offers a highly flexible approach to ontology matching, facilitating both customization and generic applicability of the matching process. In the following chapter, we will turn to the implementation of the described methodology within a flexible matching framework.
4 Implementation of the Methodology

Having described the different components that make up our methodology, this section presents its implementation within a flexible framework for ontology matching.

First, we introduce the schema matching tool COMA++[Do05], which served as a framework architecture for realizing our methodology (Section 4.1). We then present the implementation of our prototype COMA++(O). Section 4.2 describes how we extend the functionality of COMA++ by integrating the proposed ontology graph representation and providing adequate means for import and processing of ontologies. Based on this functionality, we realize the presented approaches for ontology matching as components of the framework. In particular, we present the implementation of different integration strategies in Section 4.3 and describe a selection of implemented matchers realizing various approaches to similarity computation in Section 4.4. Therein, we will particularly relate how we apply the previously described matching strategies for matcher configuration. Section 4.5 presents the GUI facilities that enable a user to configure the new matchers. Section 4.6 summarizes the chapter.

4.1 COMA++ — A Schema Matching Framework

In the previous chapter, we motivated and devised an ontology matching methodology that enables the use of different approaches and their flexible integration in a complex matching process. A tool that provides part of the proposed functionality is the schema matching tool COMA++[Do05]. Entirely implemented in Java, COMA++ provides a generic framework solution to the schema matching problem. In the following we give an overview of the architecture and functionality of COMA++, before evaluating how we adapted and extended the tool to realize the discussed concepts for matching of ontologies.

4.1.1 Architecture and Functionality Overview

This section gives a brief overview of the architectural components of COMA++. We will refer back to those components in the remainder of this chapter when addressing the implementation and integration of our methodology.

Figure 4.1 shows the architecture of COMA++, consisting of five components. A central Repository is provided to store schemata and ontologies, as well as any produced or imported mappings. Schemata and mappings can be loaded to the Schema Pool and Mapping Pool, respectively, where they are managed and can be accessed for matching-related operations. The Schema Pool further provides import functionality to parse and transform different schema representations into a generic internal representation. The Mapping Pool allows to modify and utilize available mappings.

The components concerned with the actual matching process are the Match Customizer and the Execution Engine. The former provides a Matcher Library containing predefined and user-defined matchers. Those matchers can in turn be used to configure different generic Match Strategies, which determine the general conduct of the matching process. Once configured, a matcher can be saved to the repository and used for matching of schemata. Matchers are executed by the Execution Engine, where the match process takes place as an iterative process of element identification, that is, the determination of elements used as input to the matcher, the actual matcher execution, that is, the computation of similarities based

\[\text{For the sake of brevity, we will refer only to schemata in the following, although the system supports basic handling of ontologies.}\]
on the current matcher configuration, and similarity combination.

The system is complemented by an extensive GUI, which, amongst various other functionalities, enables
the configuration of matchers, loading and saving of schemata and mappings from and to the repository,
and editing of obtained mappings.

Matcher Library As mentioned above, COMA++ provides a library of predefined matchers. Those
include hybrid matchers implementing string similarity measures such as N-Gram, Suffix, and Edit-
Distance, pattern-based matchers, and a Datatype matcher, which compares datatypes based on user-
declared datatype compatibility values. Those matchers implement algorithms commonly applied in IR
and schema matching and can be applied similarly in the context of ontology matching. Furthermore,
the library comprises several combined matchers working on the element- or structure-level, such as the
Name matcher, which compares entity labels using a combination of several string matchers or the Chil-
dren, Parents, and Siblings matchers, which compare entities based on their respective neighbouring
entities in the hierarchy.

4.1.2 The Matching Process in COMA++

COMA++ provides a generic matching process, which can be configured flexibly. A user can combine
different hybrid matchers (i.e., matchers implementing basic similarity measures) and composite match-
ers to integrate various available information about schema elements in the matching process. We will
now give a short account of how the matching is conducted in COMA++ using a composite matcher.

Figure 4.2 illustrates the COMA++ matching process, preceded by a Schema Manipulation phase and
succeeded by a Mapping Manipulation phase. Before we can execute a matching process, we first need
to parse and import the schemata to the repository, where they are represented in the form of directed
cyclic graphs. Once the schemata are imported, the user can load them to the Schema Pool and select
them for a matching process. After the matching has been carried out, the user can correct the resulting
mappings, store them persistently in the repository, and further utilize them in the Mapping Pool.

The match process itself involves three consecutive steps, which are conducted in one or several itera-
tions, depending on the chosen Match Strategy:
4.1. COMA++ — A SCHEMA MATCHING FRAMEWORK

Figure 4.2: The matching process in COMA++ (adapted from [Do05])

1. **Element Identification** Based on the given matcher configuration, the schema elements relevant for the matching, that is, those elements for which similarity values shall be computed by the current matcher, are determined.

2. **Matcher Execution** The matchers that have been selected for the matching are executed to compute individual similarity values for all input element pairs. This yields a similarity cube consisting of one similarity matrix for each constituent matcher holding the computed similarities for all element pairs. Thus, a matcher execution with $k$ constituent matchers, $m$ elements in the first schema and $n$ elements in the second schema yields a $k \times m \times n$ cube of similarity values.

3. **Similarity Combination** In the last step, a combined result is derived from the individual matcher results for each element pair. For this purpose, the matcher-specific similarity values are first aggregated into a single similarity value for each element pair, thus leading to a similarity matrix containing $m \times n$ similarity values. Then, the elements are ranked according to their similarity, with respect to the elements of either of the two input schemata. Finally, those element pairs which are suggested as plausible mappings according to a particular strategy (e.g., choosing the pair with the Maximum similarity) are selected. Finally, for combined matchers, the similarity values that have been computed for sets of element constituents or related elements must be combined into a final similarity result.

Note that we can roughly relate the COMA++ matching process to the generic process description introduced in Section 2.5.1, namely lift and normalization, computation of similarities, semantic bridging, execution, and postprocessing of the obtained mappings. For more detailed information on the matching process of COMA++, please refer to [Do05].

4.1.3 COMA++ as Basis for Flexible Ontology Matching

Resuming our discussion of a flexible framework approach to the ontology matching problem, we consider COMA++ a good basis for implementing our methodology. First, apart from the basic similarity computation and mapping extraction functionality, the system already provides part of the functionality we discussed in the context of integration and configuration of the matching process. Second, it comes with a large number of basic and composite matchers we can use as a basis for the development of new, ontology-based matching approaches. Third, the tool has to its name a good record of performance in the
field of schema matching, and was even applied as a generic solution to the ontology matching problem. A comprehensive evaluation of the tool’s performance can be found in [Do05].

COMA++ was devised for the matching of database schemata and therefore — although providing basic support for ontologies — does not leverage the wide spectrum of semantics that can be drawn from ontology entity descriptions. This suggests there is some potential for improving the quality of the generic ontology matching results. When integrating the proposed ontology matching methodology with COMA++, we will also be able to evaluate the impact of different aspects of our methodology, such as the distinction between class and property semantics, on the obtained matching results.

In the implementation of COMA++(O), we can reuse and incorporate the architectural infrastructure and the general matching process of COMA++ to a large extent. This section provides a short overview of the issues we must tackle in order to realize the presented ontology matching methodology in the prototype COMA++(O).

As pointed out, COMA++ has been developed for schema matching. Thus, its basic import and repository components, as well as the implemented matchers and the execution engine that performs the matching are geared towards the processing of schema information. In particular, the chosen internal representation and the system matchers are restricted to hierarchically structured information, ignoring other forms of semantic information. Therefore, we first need to adapt and extend the basic framework functionality of COMA++ as a prerequisite for implementing ontology-specific matching approaches:

The internal representation of ontologies is extended in compliance with the typed directed graph structure presented in Section 3.3.2, yet shall remain compatible with the current match processing functionality.

Parsing functionality is provided to allow the system to parse all relevant ontology features and make them available for further processing.

Import and repository functionality is extended in compliance with the internal ontology representation. Semantic information extracted from ontologies must be stored in respective relational data structures, from which they can then be loaded into the typed directed graph structure at run time.

Once we have adapted the basic framework functionality to allow proper handling of the semantic features of ontologies, we can implement the dedicated matching approaches (matchers) and additional integration functionality. This affects the following aspects of the system:

The match strategies are complemented with further strategies to provide for the discussed integration functionality, thus enabling the use of dependent matchers and automatic configuration of the matching process.

The matcher library is extended with additional matchers leveraging the discussed ontological description features. The implementation of new matchers implies additional strategies for element identification and similarity computation. Element identification must take into account the different types of ontology elements (such as classes and properties) and is provided by corresponding utility functions in the implementation of the ontology graph. The measures for similarity computation are mainly extended by the discussed lexical similarity measure utilizing WordNet, as described in Section 3.4.4.

In the next section we describe how we realize the described extensions to the basic framework functionality so as to integrate the ontology graph representation. Subsequently, we present the implementation of the ontology matching functionality, building on top of the extended representation.
4.2 Integrating the Ontology Graph Representation

COMA++ offers generic support for parsing and processing of ontologies. However, since its internal representation format is based on a simple directed graph structure, which can represent only one relation type, all semantic relationships between ontology elements are generally reduced to hierarchical (is-a) relations. While subclassOf and subPropertyOf constructs can be represented in the resulting schema graph in a semantically sound way, we lose at least part of the semantics of all other relations, such as those semantics inherent in a disjointWith construct or a domain declaration. Therefore, we need to extend the basic framework functionality to provide the prerequisites for realizing our methodology, that is, to account for the entire semantics of ontologies in the matching process.

In this section, we first discuss the implementation of the ontology graph representation discussed in Section 3.3.2 and then briefly present how we adapted the COMA++ parsing and importing functionality in order to account for the new internal representation.

4.2.1 Implementation of the Ontology Graph Representation

For its internal graph representations, COMA++ uses OpenJGraph [Sal02]. This is an open source API for creation and manipulation of different types of graphs, providing graph algorithms and basic visualization functionality. COMA++ extends the directed graph implementation of the API, adding utility functions for retrieval of related elements during the matching process.

In Section 3.3.2 we sketched out an ontology graph representation, which uses different vertex and edge types to represent ontology semantics. Correspondingly, we extend the hitherto applied directed graph implementation to a typed directed graph, that is, a graph representation that is aware of different vertex and edge types. Basically, for each feature we want to exploit in the matching process, that is, for each ontology relation construct we want to represent, a separate set of edges is introduced. The graph implementation and handling of graphs in the system is thus extended as follows:

Typed vertices and edges In compliance with Section 3.3.2, different vertex classes are introduced to represent different types of ontology elements. Figure 4.3 shows the class TypedVertexImpl with subclasses ClassVertex, PropertyVertex, RestrictionVertex, and BooleanVertex. The class TypedDirectedEdgeImpl is used to represent the directed edges denoting different relation types. Since we do not require special functionality for different edge types, we use a simple identifier to distinguish different relations.

Graph traversal We need to extend the basic graph algorithms for accessing adjacent vertices in the graph in such a way that the type of an edge is considered. All vertex data of a graph is held in a HashMap containing an instance of the data structure TypedVertexData for each vertex, as illustrated in Figure 4.3. For a given vertex, TypedVertexData provides access to all incoming and outgoing edges of different relation types, thereby providing the basic functionality for traversing the graph.

Utility functions are implemented for the different vertex types so as to enable convenient retrieval of vertices linked through edges of different types. That is, depending on the type of element represented by the vertex, we can access the adjacent vertices representing its related elements.

Accessing Related Elements and Constituents As stated above, the vertex implementations each provide a number of utility functions to access related elements through edges of different types. Listing 4.1 shows some methods provided by class vertices to access the related elements of a class element. For example, its parent classes, properties, and the restrictions that apply on the class can be accessed through edges of type IS_A, PROPERTY, and APPLIES_TO, respectively.
Constituents such as labels, comments, datatypes or property characteristics do not involve or refer to other elements in the graph, but rather represent some self-contained information about a certain aspect. Therefore, in keeping with COMA++, they are not represented as vertices in the graph structure, but included as attributes of the corresponding vertex. As pointed out above, class and property instances are also represented as attribute information within the graph. This solution is sufficient to use basic instance based matchers implemented in COMA++, while at the same time resulting in a smaller number of vertices in the graph, which can be significant if the ontologies are populated with large numbers of instances. We can easily extend the graph implementation with further vertex and edge types, should the need arise.

### 4.2.2 Parsing, Import and Loading of Ontologies

The proposed methodology relies on a range of OWL DL features internally represented by the described ontology graph. To make this information available in the matching process, we need to provide for appropriate parsing, import and repository facilities.

The generic ontology support of COMA++ does not provide the required functionality. As mentioned
above, COMA++ takes a schema-based view on ontologies. Some features of OWL are not supported by the COMA++ parser. Others are parsed but their semantics are not taken into account when importing them into the database, because the internal representation is not powerful enough. Therefore, we extend the repository import and loading functionality to comply with the previously described ontology graph implementation.

For parsing OWL ontologies, COMA++ uses the OWL API [OWL07]. This is an open source Java implementation of OWL, providing a representation of OWL ontologies, as well as an interface to inference and validation systems. We use OWLRDFParser to parse the ontology files. The parser returns an instance of OWLOntology, which provides access to all element descriptions of the ontology. The COMA++(O) importer loads this ontology into the database, processing each of the parsed classes, relations, class axioms, property axioms, and individuals of the ontology. Information about entities (named classes and properties), anonymous elements (restrictions and boolean combinations), and the corresponding relations are stored in such a way that their semantics are represented in the database in compliance with the described ontology graph structure. As discussed before, we store instance data of classes and properties in an attribute of the respective class or property element.

When loading the stored ontology descriptions into memory for match processing, we convert the retrieved data into typed vertices and edges, which build up the ontology graph representation.

Apart from the information inherent in the ontology element descriptions, COMA++(O) also makes use of auxiliary information from the lexical database WordNet. In Section 4.4.2 we describe how the system accesses this information for the corresponding matcher component.

Having realized the basic functionality for importing ontologies and accessing their semantics, we now turn to the implementation of the discussed integration and dedicated ontology matching functionality.

### 4.3 Matching Strategies for Integration of Individual Matchers

To provide the integration functionality described in Section 3.5, we equip the system with two additional Match Strategies, namely the DependentMatcher and the StrategyMatcher. Furthermore, we introduce an extended propagation scheme, the PropagationMatcher, which applies the previously discussed propagation of similarities between ontology elements, but allows to conduct this propagation iteratively.

Complementing the new Match Strategies, we provide repository functionality to store and load instances of DependentMatcher, StrategyMatcher, and PropagationMatcher. The user can configure instances of the Match Strategies through additional GUI facilities, which we briefly address in Section 4.5.

#### 4.3.1 DependentMatcher

As discussed in Section 3.5.3, the DependentMatcher enables matcher execution depending on a given threshold constraint. We realize this functionality as a Match Strategy in the COMA++ framework architecture.

For a DependentMatcher, we specify a baseMatcher and a depMatcher as constituent matchers, as well as a threshold and a flag above, which specify the dependency constraint setting. Listing 4.2 shows how a DependentMatcher computes the similarity cube based on its constituent matchers.

```java
// computation of similarity cube
float[][][] computeSimCube(List elemsA, List elemsB) {
    // compute element similarities for baseMatcher and depMatcher
    simCube[0] = baseMatcher.computeSimMatrix(elemsA, elemsB);
```
First, the method `computeSimMatrix` is executed on `baseMatcher` to compute the similarity matrix containing all element pair similarities, which is then inserted in the similarity cube. The similarity values for `depMatcher` are computed similarly. However, they are only inserted into the similarity cube if the specified threshold constraint is satisfied. The method `setDependentSimMatrix` illustrates how for each element pair `a`, `b` the similarities of `baseMatcher` are compared to the threshold value. If the threshold constraint is met, the similarity value `depSimMatrix[a][b]` computed by `depMatcher` is included in the `simCube`. Otherwise, the final similarity is determined solely from the results of `baseMatcher`. To this end, the similarity of `baseMatcher` is inserted again in `simCube`, which is necessary so that similarities are not scaled down when aggregating the similarity cube.

### 4.3.2 StrategyMatcher

We discussed the idea of automatic matcher configuration and the `StrategyMatcher`, which realizes this concept in our methodology, in Section 3.6.1. Like `DependentMatcher`, we implement `StrategyMatcher` as a Match Strategy of the framework. The conducted matching process differs from the standard composite strategy in that the matchers that shall be applied as constituent matchers in the matching process must be selected before the matching is started. For this purpose, a `StrategyMatcher` is configured with sets of `defaultMatchers`, `labelMatchers`, and `structureMatchers`, as well as thresholds `tLabel` and `tStructure`.

Listing 4.3 shows how the method `selectStrategyMatchers()` is applied on matcher `strategy` before the similarity matrix for `elemsA` and `elemsB` is computed in the common way.
The method \texttt{selectStrategyMatchers()} uses the values \texttt{simLabel} and \texttt{simStruct}, which represent the overall task characteristics with respect to the labels and structure of the considered ontologies. Those values are computed before the actual matching process is triggered and can be accessed through a central \texttt{Manager} component. Based on the retrieved similarity values and the thresholds \texttt{tLabel} and \texttt{tStruct}, the \texttt{StrategyMatcher} then assembles its set of constituent matchers. In particular, while the \texttt{defaultMatchers} are included as constituent matchers by default, the \texttt{labelMatchers} and \texttt{structureMatchers} are only included if the respective threshold \texttt{tLabel} or \texttt{tStruct} is exceeded, respectively.

Once the constituent matcher set is determined, the standard matching routine is executed on \texttt{strategy}.

### 4.3.3 PropagationMatcher

The functionality of our methodology as discussed in Chapter 3 extends the similarity computation functionality of COMA++ by including a variety of ontology-specific features in the matching process. Thereby, we can make use of more distinguished similarity propagation strategies considering different relations (edge types in the ontology graph) for propagation of similarities. However, this propagation scheme is still very restrictive, since it allows propagation to take place based on only one related element, and only once per matcher.

Therefore, we additionally implement the \texttt{PropagationMatcher} strategy to facilitate extended propagation. It can perform several iterative propagations using different types of related elements. This is closer to the advanced similarity propagation schemes applied in other systems such as RiMOM [YLT06], and follows the idea of \textit{Similarity Flooding} proposed in [MGMR02]. However, we only perform a certain number of iterations, while the mentioned approaches apply a fix-point algorithm, that is, propagation is applied until a given stopping criterion is met.

Listing 4.4 illustrates how the propagation functionality is implemented in the \texttt{PropagationMatcher} Match Strategy. We can configure the matcher to perform \texttt{n} iterations, specifying a set of \texttt{propagationElements}, that is, related elements such as children, parents, or properties which shall be considered for similarity propagation during each iteration.
The strategy differs from the standard matching process in that the similarity matrix computed by `computeSimMatrix()` is used as a seed value to the `propagateSim()` method, which iteratively propagates the computed similarities to a number of related elements. For each element pair, `propagateSim()` calls `computePropagationSim()` to compute the propagated similarity value with respect to some related propagation element `ele` of `a` and `b`, such as the children, parents, or properties of `a` and `b`. For this purpose, `computePropagationSim()` determines all related elements of the given type and computes `propagatedSim[a][b]` as the combination of their similarities. Average is used by default to combine the propagated similarity values from several related elements. Finally, the method `adjust` uses the similarity matrix `propagated` to adjust the similarities in the `seed` matrix by a given factor `f`. This propagation scheme can be conducted for several `propagationElements` in one pass and is iterated `n` times.

4.4 Extending the Matcher Library

This section introduces the additional system matchers we provide, building on top of the ontology graph representation functionality and exploiting the new match strategies. The matchers are constructed applying the default matcher configuration facilities of COMA++, which provide a valuable service leaving us with only minimal implementation effort.

For configuring a matcher in the COMA++ framework — the functionality can be used as it stands in COMA++(O) — we must specify the particular feature that shall be used for similarity computation and the similarity measure(s), that is, the matchers we want to apply on the given feature. Additionally, we can determine how the matcher-specific values are aggregated and the mappings are selected from the final similarity result. Moreover, we need to specify additional parameters when using the new match strategies. The described configuration aspects relate to the formal discussion of element descriptions, ontology element similarity, and integration functionality in Section 3.3, Section 3.4.4, and Section 3.5, respectively.

Please refer to [Do05] for further information on the general implementation of matcher components.

As described in Section 4.1.1, the COMA++ matcher library already contains various matchers implementing basic string similarity measures and different element-level and structure-level matchers. We can reuse those matchers for configuring the new ontology-specific matchers. For example, for the composite matchers presented below, we use the COMA++ matchers `Name`, `Datatype`, and `Instance`, which compare entities based on their name, datatype, and instance data, respectively. Additionally, we adapt the COMA++ structure-level matchers such as `Children`, `Parents` and `Siblings` in compliance with the semantics of the implemented ontology graph representation.

In the following, we describe the ontology-specific matchers implemented according to the discussion of ontology element similarity in Section 3.4.4.
4.4. EXTENDING THE MATCHER LIBRARY

4.4.1 Basic Matchers

Table 4.1 gives an overview of implemented hybrid matchers, that is, matchers that apply some basic similarity measure working on (possibly several) element constituents. For each matcher, the table lists the element type it applies to and the constituent feature which is considered. We also indicate the applied similarity measure and utilized auxiliary information, if any. In the following, we refer to most of the matchers only briefly, as the underlying ideas have already been discussed in Chapter 3, and the implementation (construction) of the individual matchers follows the pattern laid out above.

<table>
<thead>
<tr>
<th>Matcher</th>
<th>Applies to</th>
<th>Feature</th>
<th>Measure(Auxiliary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElementType</td>
<td>All</td>
<td>Element type</td>
<td>$m_{\text{comp}}$ (Compatibility)</td>
</tr>
<tr>
<td>TaxRel(2)</td>
<td>Labels</td>
<td>Tokens</td>
<td>$m_{\text{taxrel} \text{length}}(2)$; (WordNet)</td>
</tr>
<tr>
<td>StructureStat</td>
<td>Class</td>
<td>Structure statistics</td>
<td>$m_{\text{num}}$</td>
</tr>
<tr>
<td>RelationStat</td>
<td>Class, Property</td>
<td>Relation statistics</td>
<td>$m_{\text{num}}$</td>
</tr>
<tr>
<td>PropFunct</td>
<td>Property</td>
<td>Functional</td>
<td>Equality</td>
</tr>
<tr>
<td>PropSymm</td>
<td>ObjectProperty</td>
<td>Symmetric</td>
<td>Equality</td>
</tr>
<tr>
<td>PropTrans</td>
<td>ObjectProperty</td>
<td>Transitive</td>
<td>Equality</td>
</tr>
<tr>
<td>RestrType</td>
<td>Restriction</td>
<td>Restriction type</td>
<td>$m_{\text{comp}}$ (Compatibility)</td>
</tr>
<tr>
<td>RestrCard</td>
<td>Restriction</td>
<td>Cardinality</td>
<td>$m_{\text{num}}$</td>
</tr>
<tr>
<td>CombOperator</td>
<td>Combination</td>
<td>Operator</td>
<td>Equality</td>
</tr>
</tbody>
</table>

Table 4.1: Hybrid matchers

- **ElementType** implements the comparison of element types (e.g., Class, ObjectProperty, DataProperty) based on a user-defined compatibility table. As mentioned before, the ElementType matcher is integrated in the matching process by default in the current implementation. This way, using an adequate compatibility table, we enforce a partial separation of ontology types, where classes are considered separate from properties, but object and data properties are considered potential mapping pairs. That is, any matcher that is executed in COMA++(O) is applied only on such element pairs where the element type similarity as determined by ElementType is sufficiently high.

- The TaxRel matcher applies the measure $m_{\text{taxrel} \text{length}}(m)$, leveraging taxonomic information from the lexical database WordNet. We will describe it in more detail in Section 4.4.2.

- **StructureStat** and **RelationStat** compute the similarity of entities based on their structural and relation statistics, respectively. **StructureStat** computes similarities of entities based on a vector capturing their depth within the class hierarchy, and numbers of subsumed and subsuming entities. Although it works analogue to the Statistics matcher implemented in COMA++, its semantics are different, because it only considers purely structural statistics, whereas Statistics included relation statistics (due to the lack of semantic distinction in COMA++). **RelationStat** on the other hand compares relation statistics, that is, the numbers of properties of a class, or the number of domain and range classes of a property, respectively. We apply numeric measures $m_{\text{num}}$ such as Euclidean distance to compute the similarity of statistical data.

- The matchers PropFunct, PropSymm, and PropTrans compare properties based on their characteristics, that is, whether they are functional, symmetric, or transitive. If two properties exhibit the respective characteristic, they are assigned a similarity of 1, otherwise 0.

- **RestrType** and **RestrCard** compute restriction element similarities based on their type and cardinality information, if applicable.

---

2In keeping with [Do05], we refer to those matchers as hybrid, since they combine different approaches for computing the similarity between, for instance, two strings. At the same time, we want to emphasize that we regard them as basic matchers as opposed to the composite, more complex matchers.

3In contrast to Chapter 3 we distinguish more element types in the implementation of the prototype.
• **CombOperator** compares boolean class combinations based on their operator type, returning a similarity of 1 if the operator types are equal.

### 4.4.2 The Lexical Matcher

We will now briefly describe the matcher TaxRel, which relies on auxiliary information from the lexical database WordNet. We first describe how we access the required information from WordNet and then relate how we utilize it to compute label similarities.

**Using WordNet as Auxiliary Lexicon** In WordNet, nouns, verbs, adjectives and adverbs are assigned to so-called *synsets* (sets of synonyms), according to the different meanings (*senses*) they have. Words can be assigned to several synsets. Synsets are arranged within a taxonomy. Thus, for each sense of a word, we can determine the corresponding synset and on this basis find linguistically related words. For example, we can access the synonyms of a word, which are simply the members of the current synset. Moreover, based on relations between synsets in the WordNet taxonomy, we can, for example, determine hypernyms (superconcepts) and hyponyms (subconcepts).

To access the WordNet database, COMA++(O) makes use of the Java WordNet Library (JWNL) [JWN07]. This API provides useful functionality to discover words and their synsets, as well as relations between synsets in the database.

**Matching Based on Taxonomic Information** The matcher TaxRel computes similarities of concept labels *labelA* and *labelB* based on their relation within the WordNet taxonomy. Listing 4.5 describes how the similarity measure $m_{\text{taxrel\_length}}(max)$ applied by TaxRel works.

```java
float computeTaxRelSim(String labelA, String labelB, int max) {
    // lookup indexWords for the two labels
    IndexWord indA = Dictionary.lookupIndexWord(labelA);
    IndexWord indB = Dictionary.lookupIndexWord(labelB);
    // get all paths between pairs of associated synsets of indA, indB
    List paths = getTaxRel(indA, indB, int max);
    int minimalLength = getMinimalPathLength(paths);
    // compute similarity based on minimalLength
    // with respect to maximal considered path length max
    return computeRelativeSim(minimalLength, max);
}
```

Listing 4.5: Similarity computation leveraging taxonomic information

First, we have to find entries in the WordNet database that correspond to the given labels (in fact, we first split the labels into tokens). We use the JWNL Dictionary to look up the two labels in the database. If we find an entry — an *IndexWord* — for both *labelA* and *labelB* in the database, the method getTaxRel() searches for relations between the *synsets* associated with *indA* and *indB*. The length of the relation path through the taxonomy is restricted to *max*. Next, getMinimalPathLength() retrieves the shortest path of all discovered relationship paths and returns the *minimalLength*. Finally, computeRelativeSim() computes the similarity between the two labels by relating the minimal discovered path length to the specified maximum path length *max*. We set *max* to 2 by default, that is, only paths up to the length of 2 are considered in the similarity computation.
4.4.3 Composite Matchers

Table 4.2 gives an overview of implemented composite matchers, characterizing for each matcher the kind of description feature it utilizes, and the applied constituent matchers. Unless stated otherwise, we use *Average* as strategy both for the *combination* of feature value similarities and *aggregation* of constituent matcher similarities, since for most matchers we consider the constituent similarities equally important.

Note that the indicated constituent matchers reflect default settings. The library can easily be extended by matchers composed of arbitrary combinations of other matchers, provided that they are applicable on the selected feature. As mentioned, matcher configuration is a straightforward process due to the generic functionality of COMA++, which can be leveraged completely in COMA++(O).

Below, we provide a brief description for each of the listed matchers. In the next section we will refer to three special configurations — *Name, TaxRel, StatPropagation*, and *Strategy* — in more detail. They are instances of the previously described Match Strategies *DependentMatcher, PropagationMatcher*, and *StrategyMatcher*.

<table>
<thead>
<tr>
<th>Matcher</th>
<th>Applies to</th>
<th>Feature</th>
<th>Constituent matchers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name_TaxRel</td>
<td>Class, Property</td>
<td>Names</td>
<td>Name, TaxRel(Below 0.6)</td>
</tr>
<tr>
<td>Children</td>
<td>Class</td>
<td>Children</td>
<td>Name, StructureStat</td>
</tr>
<tr>
<td>Parents</td>
<td>Class</td>
<td>Parents</td>
<td>Name, StructureStat</td>
</tr>
<tr>
<td>Siblings</td>
<td>Class</td>
<td>Siblings</td>
<td>Name, StructureStat</td>
</tr>
<tr>
<td>Range</td>
<td>Obj</td>
<td>Property</td>
<td>Range</td>
</tr>
<tr>
<td>Domain</td>
<td>Property</td>
<td>Domain</td>
<td>Name, StructureStat</td>
</tr>
<tr>
<td>Restriction</td>
<td>Class, Property</td>
<td>Restrictions</td>
<td>RestrType,RestrCard</td>
</tr>
<tr>
<td>PropertyChar</td>
<td>Class, Property</td>
<td>Properties</td>
<td>PropFunct,PropSymm,PropTrans</td>
</tr>
<tr>
<td>Property</td>
<td>Class</td>
<td></td>
<td>Range, Datatype, Domain, Restriction,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PropertyChar</td>
</tr>
<tr>
<td>ElementStat</td>
<td>Class, Property</td>
<td>Self</td>
<td>StructureStat,RelationStat</td>
</tr>
<tr>
<td>Equivalent</td>
<td>Class, Property</td>
<td>Equivalents</td>
<td>Name</td>
</tr>
<tr>
<td>Disjoint</td>
<td>Class</td>
<td>Disjoint</td>
<td>Name</td>
</tr>
<tr>
<td>CombOperand</td>
<td>Combination</td>
<td>Operands</td>
<td>Name, StructureStat</td>
</tr>
<tr>
<td>LexPropagation</td>
<td>Class,Property</td>
<td>Propagation</td>
<td>Name, Instance,Comment</td>
</tr>
<tr>
<td>StatPropagation</td>
<td>Class,Property</td>
<td>Subsumed</td>
<td>ElementStat,Instance,Comment</td>
</tr>
<tr>
<td>Strategy</td>
<td>Class,Property</td>
<td></td>
<td>$M_{default},M_{structure},M_{label}$</td>
</tr>
</tbody>
</table>

Table 4.2: Composite matchers

- *Name_TaxRel* is an instance of the *DependentMatcher* strategy, combining *Name* and *TaxRel* based on a dependency threshold. For *Name_TaxRel*, we use the *Max* aggregation strategy, since we assume that only one of the involved matchers yields a high similarity, which shall be the final similarity value. In Section 4.4.4 we will refer in more detail to the implementation, respectively configuration, of this matcher.

- *Children*, *Parents* and *Siblings* compare classes based on their children, parent, and sibling classes, respectively. Those hierarchy-related matchers adapt the configurations used in COMA++, restricting the use on class elements. For computing similarities between the related classes, *Name* and *StructureStat* are used.

- Correspondingly, *Range* and *Domain* compare properties based on their ranges and domains, respectively.

- The *Restriction* matcher computes similarities of classes or properties based on the type and cardinalities of the restrictions that apply to or restrict them, respectively.
• PropertyChar combines the matchers that compare property characteristics. We use the Max strategy for aggregation, since we consider it sufficient that one property characteristic matcher (PropFunct, PropSymm, or PropTrans) discovers a matching characteristic.

• Property uses several of the above-mentioned matchers to compute class similarities based on their properties, leveraging their range and domain classes, datatypes, restrictions that are defined on them, and property characteristics.

• ElementStat combines the two statistic matchers to exploit all available statistics we can access for class and property elements.

• Equivalence and disjointness information can be leveraged using the Equivalent and Disjoint matchers, which compute similarities of equivalent or disjoint classes, respectively, based on their names.\footnote{Note that we could not test these matchers due to the lack of appropriate test data.}

• Boolean class combinations are compared based on their operands by CombOperand.

• LexPropagation and StatPropagation use the PropagationMatcher Match Strategy to propagate linguistic, statistical, and instance-based similarities through the hierarchy and between classes and properties. We describe the matcher StatPropagation in some detail in Section 4.4.4.

• The configuration of Strategy as an instance of StrategyMatcher will also be described in Section 4.4.4.

Composite matching process: Property We will now briefly exemplify the matching process that is carried out when using the composite matcher. The matcher Property is an example for a rather complex composite matcher, built of various constituent matchers that are in turn composed of other matchers. Property compares class elements of an ontology based on their properties. The similarity values for the properties are in turn computed based on the similarity results produced by five matchers. The Domain matcher computes similarities based on different features of the domain classes of the property, as indicated in Table 4.2. For object and data properties, the Range and Datatype matchers compute similarities of the range classes and datatype of the property, respectively. Furthermore, a Restriction matcher and a PropertyChar matcher, which leverage the type and cardinality of restrictions that restrict a property, and the characteristics of properties, respectively, are applied. Finally, the Property matcher aggregates the five constituent matcher similarity results using an Average operator, since we consider the different features equally important.

Finally, the similarities thus obtained for the properties are combined in order to yield the similarities of the class elements that are compared. Based on the resulting similarities, those class pairs with the highest similarity are selected as mappings.

4.4.4 Applying the Matching Strategies

This section describes some of the previously presented matchers in more detail. In particular, we relate how Name_TaxRel, StatPropagation and Strategy are configured as instances of the new matching (integration) strategies of COMA++(O).

DependentMatcher Configuration: Name_TaxRel We introduced the concept of dependent matching as an advanced integration functionality in Section 3.5.3 and described its implementation in Section 4.3.1. As a possible application case for such a matcher, we described a matching task where we want to give one constituent matcher priority over another. This concept is applied in the matcher Name_TaxRel. Table 4.3 shows how the matcher is configured.
4.4. EXTENDING THE MATCHER LIBRARY

<table>
<thead>
<tr>
<th>Matcher</th>
<th>$matcher_{base}$</th>
<th>$matcher_{dep}$</th>
<th>Aggregation</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name_TaxRel</td>
<td>Name</td>
<td>TaxRel(2)</td>
<td>Max</td>
<td>$sim_{base} &lt; 0.6$</td>
</tr>
</tbody>
</table>

Table 4.3: DependentMatcher configuration example

Name_TaxRel applies as constituent matchers Name and TaxRel(2). In particular, the Name matcher is specified as the base matcher, whereas the TaxRel(2) matcher is declared dependent on the results of Name. That is, as described above, the similarity values computed by TaxRel(2) for an element pair $e_a, e_b$ only contribute to the overall similarity result if the threshold constraint is satisfied, that is, if $sim_{base}(e_a, e_b) < 0.6$. The motivation for this configuration is that often no sufficient taxonomic relationship can be retrieved from WordNet, even when the computed name similarity is high. We want to avoid that the similarity values computed by TaxRel(2) are included in the final result whenever Name yields a sufficiently high similarity, that is, $sim_{base}(e_a, e_b) \geq 0.6$. On the other hand, if name similarity is low, we want to try and see if there is a taxonomic relationship (e.g., synonymy) that yields a high similarity value $sim_{dep}(e_a, e_b)$. We aggregate the similarities using the Max operator, because we assume that often at most one of the matchers yields a high similarity.

StrategyMatcher Configuration: Strategy

Strategy is an instance of StrategyMatcher. The configuration of Strategy, as shown in Table 4.4, was determined based on our experience from different matching tasks from the OAEI benchmark series. The matcher is configured with three matcher sets $M_{default}$, $M_{structure}$, and $M_{label}$. The first contains matchers that are largely independent of label and hierarchic information. In particular, RelationStat and Instance compare entities solely based on their relation statistics and instance data, respectively. $M_{structure}$ contains the matchers StructureStat and StatPropagation, which leverage mainly structural information, and $M_{label}$ correspondingly contains the label-based matchers Name and LexPropagation. The similarity thresholds are set to $t_{struct} = 0.4$ and $t_{label} = 0.2$.

<table>
<thead>
<tr>
<th>Matcher</th>
<th>$M_{default}$</th>
<th>$M_{structure}$</th>
<th>$M_{label}$</th>
<th>Thresholds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Comment, Relation-Stat, StatPropagation, Instance</td>
<td>StructureStat, StatPropagation</td>
<td>Name, LexPropagation</td>
<td>$t_{struct} = 0.4$ and $t_{label} = 0.2$</td>
</tr>
</tbody>
</table>

Table 4.4: StrategyMatcher configuration example

When Strategy is executed, the actual matching process is preceded by a constituent matcher selection according to the threshold values and the task specific similarity values $sim_{struct}$ and $sim_{label}$, which are computed for the two selected ontologies when the matching is triggered. For example, if $sim_{struct} = 0.3$ and $sim_{label} = 0.9$, the constituent matchers $M$ of Strategy are set to $M = M_{default} \cup M_{label}$. As described before, once the constituent matchers are selected, the standard similarity computation process is started.

PropagationMatcher Configuration: StatPropagation

The StatPropagation matcher uses the PropagationMatcher Match Strategy to propagate similarities iteratively through the hierarchy and between classes and their properties. It is configured like a conventional composite matcher, but instead of specifying a feature to be leveraged for similarity computation, it allows the user to specify a number of related elements which shall be considered for similarity propagation, as well as the number of iterations to perform. Table 4.5 shows the configuration of StatPropagation.

First, StatPropagation computes the similarity values of all element pairs by applying the constituent matchers ElementStat and Instance. The resulting initial similarity values are then propagated through the hierarchy, that is, to Children and Parents, and between classes and their properties, that is, to
4. IMPLEMENTATION OF THE METHODOLOGY

<table>
<thead>
<tr>
<th>Matcher</th>
<th>Constituent matchers</th>
<th>Propagation elements</th>
<th># iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatPropagation</td>
<td>ElementStat, Instance</td>
<td>Children, Parents, Properties, Domains</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.5: PropagationMatcher configuration example

Properties and Domains. The matcher performs 4 iterative similarity propagations to the specified related elements.

4.5 Graphical User Interface

The COMA++ framework comes with an extensive GUI, facilitating a user to manage and manipulate schemata and mappings, configure matchers, and interact in the matching process [Do05]. We extended this GUI with additional configuration facilities and adapted the visualization of ontologies.

4.5.1 Visualization of Ontologies

The visualization of ontologies in COMA++(O) is based on the schema visualization functionality of COMA++. That is, ontologies are visualized in the form of a tree. Subclasses and properties are represented as leaves of the respective superclass or domain class. Information about other semantic features, such as the range of a property or restrictions on a property are included in the respective property nodes in order to keep the presentation simple.

Since the modifications of the visualization functionality are only minor, we do not discuss them further here. Please refer to [Do05] for more information about the visualization of the mapping task.

4.5.2 Configuration Facilities

We extended the existing configuration facilities in order to enable the construction of matchers that use the new Match Strategies. In the creation wizards for the DependentMatcher, StrategyMatcher, and PropagationMatcher, the user can specify the sets of constituent matchers and thresholds as required by the respective strategy.

Figure 4.4: Creating a new DependentMatcher

A DependentMatcher is configured as shown in Figure 4.4. First, the feature that shall be leveraged by the matcher (denoted here as Constituents) is specified. Then, the user chooses from the list of available matchers the base matcher Base and the dependent matcher Dep and specifies the threshold constraint by providing a threshold and stating whether the similarity result of Base should be above or below that threshold.
Figure 4.5 shows how a **StrategyMatcher** is configured by choosing the feature the matcher should utilize and then specifying the configuration strategy. To this end, the user selects from the list of available matchers three sets of **Default**, **Structure**, and **Label** matchers. Furthermore, the user can provide the threshold values for the structure and label similarities that must be exceeded in a given task in order for the respective matcher sets to be included in the matching process.

Figure 4.6 illustrates the configuration of a **PropagationMatcher**. Other than in the previously discussed configurations, the user does not specify the leveraged feature for a **PropagationMatcher**, because this is set to **SelfNode** by default (i.e., the considered element itself is taken as input to the matcher). The user selects matchers that shall be used to compute the initial similarities of the elements and then specifies a propagation strategy by selecting a list of propagation elements and setting the number of iterations to be performed.

### 4.6 Summary

This chapter presented the prototypical implementation COMA++(O), which realizes our ontology matching methodology, leveraging the framework architecture of COMA++. COMA++ provided us with extensive generic functionality for matcher construction and similarity computation, as well as basic facilities for import, storage, and visualization of ontologies. Our implementation extends the available facilities so as to account for the rich semantics of OWL DL language features and make them available in the matching process. To this end, we implemented a graph-based ontology representation and added functionality to parse, store, and load ontologies in this representation. We then complemented the system with a set of ontology-specific matchers, which implement a range of similarity measures defined in Section 3.4.4. Furthermore, we add new Match Strategies enabling more involved integration of matchers, as well as automatic configuration of the matching process.

Concluding, the framework architecture of COMA++ provided valuable service for the implementation of our ontology matching methodology, allowing us to realize our approaches with minimal implementation effort. The resulting prototype COMA++(O) leverages the basic functionality of the framework and adds support for matching of ontologies, accounting for their specifically rich semantics and the diversity of ontology matching tasks.

In the following section we evaluate the presented prototype and compare it against a number of other state-of-the-art systems for ontology matching.
5 Evaluation of the Prototype

To survey the quality of our ontology matching methodology, we performed a comprehensive evaluation of the prototype COMA++(O) against a systematic series of test cases specified for the OAEI 2006 campaign. This chapter presents the conduct and results of the evaluation.

First, we describe the general procedure commonly employed for the evaluation of schema and ontology matching results (Section 5.1). After that, we introduce the applied test base and experiment settings in Section 5.2 and 5.3. Section 5.4 presents and discusses the results of the overall evaluation, and Section 5.5 provides some insights about the influence of individual approaches applied within the methodology. In Section 5.6 we compare the obtained matching quality with that of other systems. The overall results of the evaluation are assessed in Section 5.7. In Section 5.8 we comment on identified benefits and shortcomings of the conducted evaluation, before we summarize the chapter in Section 5.9.

Before presenting the evaluation of our methodology, we want to advise that determining the quality of a matching result is a very subjective matter, since in many cases it is not even definite for a human which ontology elements constitute real correspondences. Furthermore, as already discussed in the opening of this thesis, the problem of ontology matching is highly context-dependent. We should keep those issues in mind when evaluating matching results, even when the evaluation procedure yields seemingly objective quality measures.

5.1 Evaluation Procedure and Measures

The evaluation of ontology matching results is generally conducted against a reference alignment, that is, a manually solved matching task. This reference alignment contains all mappings that are considered correct for a given matching task. By comparing a (semi-)automatically obtained matching result against this gold standard, we can partition the obtained mappings into four sets. The true positives $T_p$ are those element pairs that are both contained in the obtained matching result and in the reference alignment. Conversely, the false positives $F_p$ are incorrectly obtained correspondences. True negatives $T_n$ are correspondences that are included neither in the matching result nor in the reference alignment, while false negatives $F_n$ are those correspondences which are intended in the reference alignment but not discovered by the matcher. Based on those four sets, the quality measures Precision and Recall can be computed:

- **Precision** reflects the share of correct correspondences among all discovered ones:
  \[
  \text{Precision} = \frac{T_p}{T_p + F_p}
  \]

- **Recall** reflects the share of correct correspondences that is discovered:
  \[
  \text{Recall} = \frac{T_p}{T_p + F_n}
  \]

In order to reflect the quality of a matching result with respect to both Precision and Recall, combined measures have been devised. Amongst those, $F$-measure($\alpha$) is arguably the one most commonly used for evaluation tasks in the fields of machine learning and information retrieval.

\[
F\text{-measure}(\alpha) = \frac{\text{Precision} \times \text{Recall}}{(1-\alpha) \times \text{Precision} + \alpha \times \text{Recall}}, (0 \leq \alpha \leq 1)
\]
5. EVALUATION OF THE PROTOTYPE

By specifying a weight $\alpha$, we can vary the influence of Precision and Recall on the overall quality measure. The most common variant (which we use in this evaluation) is $F-measure(0.5)$. It reflects the harmonic mean of Precision and Recall.

$$F-measure = F-measure(0.5) = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Another combined measure, which has been developed specifically for the evaluation of matching results, is $Overall$.

$$Overall = \text{Recall} \times \left(2 - \frac{1}{\text{Precision}}\right)$$

The introduced measures all reflect in some way the quality of the matching result. However, each of them emphasizes a different aspect of that result. Interestingly, we can view those measures in terms of how much effort a user of the system still needs to invest in order to correct the discovered mappings. In this respect, a low Precision value means that many false mappings have to be removed from the result, whereas a low Recall implies that many additional mappings must be identified manually. Consequently, depending on whether we strive for a highly precise result or a result that contains a high number of intended results (but possibly also many false correspondences), we will aim at maximizing either Precision or Recall. F-measure and Overall can also be interpreted from that perspective. F-measure estimates the match quality optimistically with equal weights on Precision and Recall, always yielding positive values. Overall yields much lower values for similar Precision and Recall values, with particular sensitivity to low Precision. When Precision goes beneath 0.5, that is, the discovered mappings contain more false than correct ones, Overall assumes negative values. This sensitivity to Precision reflects the assumption that it takes more effort to remove false mappings than to add mappings that have not been discovered.

Assessment of Quality Results  As we have stated above, assessing the quality of matching results is a difficult problem, even though the presented procedure and measures suggest that we can compare results objectively.

One issue that arises when we compare the obtained alignments against the reference alignments is that of *subjectiveness*. This is an inevitable problem whenever we need to evaluate an automatic approach against manually obtained data, as are the reference mappings. Another aspect that influences the assessment of the obtained results is the choice of the quality measure. As described above, various measures highlight different aspects of the result. Therefore, we must consider which measure is the best reflection of quality in a given context.

Obviously, the cost that is incurred by different kinds of alignment corrections (i.e., removing or adding mappings) depends on many different factors. This includes, for example, the application context, the task at hand (e.g., its size), the user’s expertise in the domain, and the visualization of mappings, all of which affect to a large degree how well the user can survey and modify the discovered mappings.

For the presented evaluation, we computed Precision, Recall, and F-measure for each of the tasks. We do not want to give priority to either Precision or Recall, since we conduct an application-independent evaluation. We chose F-measure for ranking and comparing match results of different approaches and systems, thereby giving equal importance to Precision and Recall.

Finally, we want to mention that there are many other aspects we could consider when evaluating a matching methodology. For example, [Do05] points out that, in addition to the effort invested after the automatic matching (i.e., for correction), we should also consider the effort involved in the pre-match step. This includes, for example, the combination of matchers or the setting of parameters. If the pre-match effort we must invest is too high, this can reduce the value of the overall matching process, even when the obtained mappings are of very good quality.
5.2 Evaluation Test Base

To give an objective account of the quality our matching methodology provides, we carry out a comprehensive evaluation based on a systematic test series specified for the OAEI 2006 evaluation campaign. The OAEI contest has been initiated to provide researchers on the field of ontology matching with a gold standard against which they can evaluate their systems. The benchmark series published for this campaign includes a number of matching tasks featuring different ontologies to be matched against a reference ontology. Each task is accompanied by a respective reference alignment. Thus, — subject to the above-mentioned issues — the test base provides an objective benchmark for assessing ontology matching systems.

5.2.1 Test Ontologies and Task Series

The test ontologies of the OAEI benchmark series comprise both synthetic and real-world ontologies. All ontologies are described in OWL-DL and are of small to medium size, with a varying number of 15 to 68 classes and 0 to 72 properties each. The real-world ontologies include BibTex, Food, MIT, UMBC, Karlsruhe, and INRIA. The synthetic ontologies were generated from BibTex by modifying one or several ontology features. All ontologies describe the domain of bibliographic reference, except Food, which is from an unrelated domain. The task is to match each of the ontologies with the reference ontology BibTex.

The reference alignments provided with the tasks include only mappings between named classes and properties. Furthermore, they use in most cases the equivalence relation with a confidence value of 1.0 (except for some mappings of the 3xx tasks, which express sub- and superclass relationships). This is compliant with our methodology, since we also restrict the possible mappings to equivalence mappings between named ontology elements of the same type.

The test base is divided into three series of tasks, herein after denoted with 1xx, 2xx, and 3xx. In the following we briefly describe the characteristics of the series:

- Series 1xx includes three simple tasks of matching the reference ontology with itself, and with its language generalization and restriction, respectively. Task 102 requires matching the reference ontology against the unrelated ontology Food.
- Series 2xx includes the tests from 201 to 266. The task is to match the reference ontology against a variant of itself, which has certain features modified or discarded. We divide this series into two parts, since the contained tasks exhibit different degrees of modification and difficulty. We refer to those as series 2xxa and 2xxb throughout the remainder of this chapter. In series 2xxa, which comprises tasks 201-247, name and comment information remains largely unchanged in most tasks, except for tasks 201 and 202. In series 2xxb (tasks 248-266), all linguistic information is systematically suppressed, leaving only one or several out of hierarchical, property, or instance information unchanged.
- Series 3xx comprises four tasks where the reference ontology is to be matched against real-world ontologies from the domain of bibliographic reference. Those were developed independently at MIT (301), University of Maryland (302), Karlsruhe University (303) and INRIA (304). Since the heterogeneity in those tasks was not introduced by systematic modifications, they exhibit more irregular divergence of features then the tasks in series 1xx and 2xx.

Table 5.1 presents the characteristics of all 51 matching tasks. For each task, it indicates if naming (N), comment (C), hierarchy (H), instance (I), or property (P) information has been modified. It also shows the number of class and property elements, and named individuals included in the ontologies, as well as the number of expected mappings for each task. Furthermore, we include the general structural and lexical similarities $\text{sim}_{\text{struct}}$ and $\text{sim}_{\text{label}}$, which were computed as described in Section 3.6.2.
<table>
<thead>
<tr>
<th>Task</th>
<th>Target</th>
<th>Modification</th>
<th>#Class/Prop/Inst</th>
<th>#Mappings</th>
<th>$t_{struct}$</th>
<th>$t_{label}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>BibTex</td>
<td>Reference ontology</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>102</td>
<td>Food</td>
<td>Irrelevant Ontology</td>
<td>63/4/33</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>103</td>
<td>BibTex</td>
<td>Language generalization</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>104</td>
<td>BibTex</td>
<td>Language restriction</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>201</td>
<td>Var 101</td>
<td>No N</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>202</td>
<td>Var 101</td>
<td>No N, no C</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>203</td>
<td>Var 101</td>
<td>No C</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>204</td>
<td>Var 101</td>
<td>N conventions</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>0.74</td>
</tr>
<tr>
<td>205</td>
<td>Var 101</td>
<td>N synonyms</td>
<td>35/63/63</td>
<td>97</td>
<td>1</td>
<td>0.34</td>
</tr>
<tr>
<td>206</td>
<td>Var 101</td>
<td>Translated N and C</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>0.29</td>
</tr>
<tr>
<td>207</td>
<td>Var 101</td>
<td>Translated N</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>0.29</td>
</tr>
<tr>
<td>208</td>
<td>Var 101</td>
<td>N conventions, no C</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>0.74</td>
</tr>
<tr>
<td>209</td>
<td>Var 101</td>
<td>N synonyms, no C</td>
<td>35/63/56</td>
<td>97</td>
<td>1</td>
<td>0.34</td>
</tr>
<tr>
<td>210</td>
<td>Var 101</td>
<td>Translated N, no C</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>0.28</td>
</tr>
<tr>
<td>211</td>
<td>Var 101</td>
<td>No H</td>
<td>35/64/56</td>
<td>97</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>212</td>
<td>Var 101</td>
<td>Flattened H</td>
<td>31/64/56</td>
<td>93</td>
<td>0.5</td>
<td>0.89</td>
</tr>
<tr>
<td>213</td>
<td>Var 101</td>
<td>Expanded H</td>
<td>68/65/56</td>
<td>97</td>
<td>0.24</td>
<td>0.51</td>
</tr>
<tr>
<td>214</td>
<td>Var 101</td>
<td>No I</td>
<td>35/64/0</td>
<td>97</td>
<td>1</td>
<td>0.97</td>
</tr>
<tr>
<td>215</td>
<td>Var 101</td>
<td>No P restrictions</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>216</td>
<td>Var 101</td>
<td>No P</td>
<td>35/0/56</td>
<td>33</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>220</td>
<td>Var 101</td>
<td>Flattened Classes</td>
<td>27/52/56</td>
<td>72</td>
<td>0.87</td>
<td>0.8</td>
</tr>
<tr>
<td>221</td>
<td>Var 101</td>
<td>Expanded Classes</td>
<td>35/64/56</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>222</td>
<td>Var 101</td>
<td>No H, no I</td>
<td>35/64/0</td>
<td>97</td>
<td>0</td>
<td>0.97</td>
</tr>
<tr>
<td>223</td>
<td>Var 101</td>
<td>No H, no P</td>
<td>35/0/56</td>
<td>33</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>224</td>
<td>Var 101</td>
<td>No P, no I</td>
<td>35/0/0</td>
<td>33</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>225</td>
<td>Var 101</td>
<td>Flattened H, no I</td>
<td>31/64/0</td>
<td>93</td>
<td>0.5</td>
<td>0.89</td>
</tr>
<tr>
<td>226</td>
<td>Var 101</td>
<td>Expanded H, no I</td>
<td>68/65/0</td>
<td>97</td>
<td>0.24</td>
<td>0.51</td>
</tr>
<tr>
<td>227</td>
<td>Var 101</td>
<td>Flattened H, no P</td>
<td>31/0/56</td>
<td>29</td>
<td>0.5</td>
<td>0.89</td>
</tr>
<tr>
<td>228</td>
<td>Var 101</td>
<td>Expanded H, no P</td>
<td>68/0/56</td>
<td>33</td>
<td>0.24</td>
<td>0.51</td>
</tr>
<tr>
<td>229</td>
<td>Var 101</td>
<td>No H, I, P</td>
<td>35/0/0</td>
<td>35</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>230</td>
<td>Var 101</td>
<td>Flattened H, no I, P</td>
<td>31/0/0</td>
<td>29</td>
<td>0.5</td>
<td>0.89</td>
</tr>
<tr>
<td>231</td>
<td>Var 101</td>
<td>Expanded H, no I, P</td>
<td>68/0/0</td>
<td>33</td>
<td>0.24</td>
<td>0.51</td>
</tr>
<tr>
<td>232</td>
<td>Var 101</td>
<td>No N, C, H</td>
<td>35/64/56</td>
<td>97</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>233</td>
<td>Var 101</td>
<td>No N, C, I</td>
<td>35/64/0</td>
<td>97</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>234</td>
<td>Var 101</td>
<td>No N, C, P</td>
<td>35/0/56</td>
<td>33</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>235</td>
<td>Var 101</td>
<td>No N, C, flattened H</td>
<td>31/64/56</td>
<td>93</td>
<td>0.5</td>
<td>0.06</td>
</tr>
<tr>
<td>236</td>
<td>Var 101</td>
<td>No N, C, expanded H</td>
<td>68/65/56</td>
<td>97</td>
<td>0.24</td>
<td>0.03</td>
</tr>
<tr>
<td>237</td>
<td>Var 101</td>
<td>No N, C, H, I</td>
<td>35/64/0</td>
<td>97</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>238</td>
<td>Var 101</td>
<td>No N, C, H, P</td>
<td>35/0/56</td>
<td>33</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>239</td>
<td>Var 101</td>
<td>No N, C, P, I</td>
<td>35/0/0</td>
<td>33</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>240</td>
<td>Var 101</td>
<td>No N, C, I, flattened H</td>
<td>31/64/0</td>
<td>93</td>
<td>0.5</td>
<td>0.06</td>
</tr>
<tr>
<td>241</td>
<td>Var 101</td>
<td>No N, C, I, expanded H</td>
<td>68/65/0</td>
<td>97</td>
<td>0.24</td>
<td>0.03</td>
</tr>
<tr>
<td>242</td>
<td>Var 101</td>
<td>No N, C, H, I, P</td>
<td>35/0/0</td>
<td>33</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>243</td>
<td>Var 101</td>
<td>No N, C, I, P, flattened H</td>
<td>31/0/0</td>
<td>29</td>
<td>0.5</td>
<td>0.06</td>
</tr>
<tr>
<td>244</td>
<td>Var 101</td>
<td>No N, C, I, P, expanded H</td>
<td>68/0/0</td>
<td>33</td>
<td>0.24</td>
<td>0.03</td>
</tr>
<tr>
<td>245</td>
<td>Var 101</td>
<td>No N, C, I, P</td>
<td>35/64/0</td>
<td>97</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>246</td>
<td>Var 101</td>
<td>No N, C, I, expanded H</td>
<td>68/65/0</td>
<td>97</td>
<td>0.24</td>
<td>0.03</td>
</tr>
<tr>
<td>247</td>
<td>Var 101</td>
<td>No N, C, H, I, P</td>
<td>35/0/0</td>
<td>33</td>
<td>0</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table 5.1: Characteristics of the OAEI benchmark tasks
5.2.2 Contest-based Evaluation

The OAEI evaluation campaign has been initiated to provide researchers with a standard procedure for evaluating and comparing their systems. Available tasks comprise the benchmark series described above, as well as a number of other data sets that are to be matched “blindly”, that is, the mappings are not provided in advance, but after results are submitted. Those tasks were therefore unsuitable for an evaluation in the scope of this thesis. Furthermore, there are tasks that require matching of large-sized taxonomies. Those tasks were also no suitable test cases for our methodology, since they do not use the broad range of ontology description features, which were the central issue of interest in this work.

To ensure comparability of the results of different contest participants, the OAEI has set guidelines as to how the evaluation must be conducted. First, the applied algorithm must be the same for all matching tasks, that is, no customization is allowed to account for the specific characteristics of the different tasks. Furthermore, users must use as input only the two source ontologies and any generally accessible resources, such as publicly available lexica. The use of auxiliary information created especially for the evaluation, as well as information from previous mappings, is prohibited.

5.3 Evaluation Objectives and Experiment Design

In the design of our methodology we had set a high value on the flexibility of the matching process, suggesting that flexible configuration facilities enable both 1) customization of the matching process to task specific needs and 2) a generic matching process that yields good overall results irrespective of the nature of the matching task.

In the following evaluation, we want to examine whether the implemented methodology lives up to those expectations. Consequently, in the evaluation of 1), it shall be allowed that matchers are customized so as to yield the optimal matching result for a specific task. On the other hand, for evaluating the methodology with regard to 2), we assert that no manual customization is carried out, but the same configuration must be applied to all tasks. In doing so, we comply with the rules specified for the OAEI contest. This allows us to compare the obtained evaluation results for 2) with those of the contest participants.

COMA++(O) is applied on the task series in two experiment settings as described below. Additionally, we investigate some individual matching approaches to give the reader an impression of their quality and influence on the matching result.

For each matcher execution, the resulting alignments are compared directly against the provided reference mappings to compute the quality measures Precision, Recall, and F-measure. As stated above, F-measure was chosen to rank the match results.

Task-specific matcher evaluation First, we test the influence of different matcher combinations on the match quality for each task. To that end, we choose a subset of 10 matchers from the matcher library, namely Name, Comment, Children, Instance, Restrictions, Property, StructureStat, RelationStat, LexPropagation, and StatPropagation, and apply each of the $2^{10} - 1$ possible combinations as a combined matcher on the complete series. The matcher that yields the best quality for a specific task can be viewed as the matcher tailored to that respective task. Thus, we take the quality obtained from the highest-rating matcher for each task as the evaluation result for objective 1). Those results are presented in Section 5.4.1.

Generic match process evaluation The second setting evaluates the quality of a single matcher configuration being applied on all tasks of the benchmark series. We use two approaches to investigate the generic matching quality.
• **Best average matcher** First, we select from the results of the previous evaluation setting the matcher configuration where the average quality over all tasks is highest. This *best average matcher* is regarded as the optimal configuration for a generic matcher, since it solves best the variety of different matching tasks of the series.

• **StrategyMatcher application** As a second approach, we apply an instance of *StrategyMatcher* on all tasks of the series. We use the matcher *Strategy*, which is set up according to the configuration provided in Table 4.4 in Section 4.4.4. This configuration is based on our experience from the best average matcher configuration and the specific matcher configurations that performed well on individual tasks.

We assume the matching results obtained by using the best average matcher and the configured *Strategy* matcher as the evaluation results for objective 2). They are presented in Section 5.4.2.

### 5.4 Experiment Results

In this section, we present and discuss the obtained results for each of the previously described experiment settings.

#### 5.4.1 Customized Matching Evaluation

The test results presented here reflect the best matching quality we obtained for the individual tasks by applying a specific matcher configuration each. In general, the matcher combinations that yield the optimal results comprise a total of 1 to 5 constituent matchers. We now discuss the best matching quality for the tasks of series 1xx, 2xxa, 2xxb, and 3xx. The detailed results are presented in Figures 5.1, 5.2, 5.3, and 5.4, respectively. Finally, Figure 5.5 shows the best task qualities averaged over the tasks of each series, and over all tasks of the benchmark.

Figure 5.1 shows the matching qualities for series 1xx.

![Figure 5.1: Best qualities for series 1xx](image)

For tasks 101, 103, and 104, we achieve the maximum obtainable quality. It is trivial to identify all intended mappings precisely, since the names of entities in the three ontologies remain the same. The modifications enforced by language generalization and restriction in task 103 and 104, respectively, do therefore not influence the match result. We do not include the result for task 102 in our evaluation. Since no mappings are intended for this task, Precision and Recall are automatically 0.0. Disregarding the result for task 102, the average Precision, Recall, and F-measure for series 1xx is 1.0, as indicated in Figure 5.5.
Figure 5.2 presents the best obtained qualities for series 2xxa.

For more than half of the tasks of series 2xxa, we achieve the maximum quality of 1.0 for Precision, Recall, and F-measure. This is because name information remains unchanged in most tasks of the series. Since we distinguish the semantics of classes and properties, mappings between a property and a class, such as the property *proceedings* and the class *Proceedings*, are precluded a priori. Therefore, when names remain unchanged a simple name matcher is sufficient to discover all mappings with absolute precision.

The average F-measure we achieve for this series is 0.99, as indicated in Figure 5.5.

- 201: Although label information is discarded (i.e., replaced by random strings) we achieve total quality for this task, since our matchers can still leverage available structure, relation, restriction, and comment information.
- 202: In addition to label information, comments are discarded. Still, we achieve an F-measure of 0.89, leveraging the structure and relation statistics and the restrictions, all of which remain unchanged.
- 203 and 204: Label information is unchanged in task 203, thus allowing precise discovery of all mappings. The same is true for task 204, where names are only slightly modified by introducing uppercasing or underscores. The name matchers implemented in COMA++ can handle such cases.
- 205: Here, a large portion of names has been replaced by synonyms. The matchers find all but one correspondence, which is due to one source element being mapped to two target elements in the reference alignment. We applied a matcher configuration that retrieves a maximum of 1 correspondence per element. Thus, we observe a negligible decrease of Recall and F-measure to 0.99 for this task.
- 206 and 207: Names and comments were translated into French, thus deteriorating the name matcher performance. Still, we achieve almost total quality. Again, a case of a double correspondence in the reference alignment causes Recall and F-measure to decrease to 0.99.
- 208 to 210: Names have been changed by altering naming conventions, replacing them by synonyms, or translating them into French. In addition, comments have been removed. The latter causes a decrease in quality as compared to the previously discussed task. While in task 208 we still achieve an F-measure of 0.99, the quality decreases for the other tasks. Still, the matchers exploit property, restriction and instance information and yield good F-measure values of 0.94 and 0.96 for task 209 and 210, respectively.
- 221 to 228: While hierarchy, instance or property information are discarded in those tasks, name and comment information remain unchanged. Therefore, all mappings can be discovered using simple name matching.
- 230 and 231: For task 230, some classes have been flattened, that is, replaced by their components in the class structure. However, we still achieve an F-measure of 1.0 by combining name and
structural information. Task 231, where classes were expanded (i.e., replaced by several classes),
poses no problem to the name matcher since the names of the original classes remain the same.

- 232 to 240: In these tasks, hierarchy, property and/or instance information were removed or
modified. For some tasks, the class hierarchy was flattened (237, 239) or expanded (238, 240),
thereby decreasing or increasing the number of classes, respectively. This does not affect the
matching quality, since the remaining classes can still be matched based on their names.

- 241 to 247: Again, by exploiting naming information all intended matches can be discovered for
tasks 241 and 246. In task 247, the matchers find a (feasible) property correspondence, which is
however not included in the reference alignment (since all properties should be discarded in this
task). Therefore, Precision decreases to 0.97 and F-measure is 0.99 for this task.

In Figure 5.3, we present the best obtained qualities for series 2xxb.

![Figure 5.3: Best qualities for series 2xxb](image1)

This series contains more complicated tasks, since name and comment information is discarded in all of
them. Therefore, we can only use hierarchy, relation, restriction and instance information, if given. Still,
using propagation of similarities based on this information, the resulting qualities are good. As indicated
in Figure 5.5, we achieve an average F-measure of 0.75 for this series.

- 248 to 250: Additionally to naming and comment information, one out of hierarchy, instance,
or property information is removed. Consequently, the matchers can only leverage little remaining
information. Still, exploiting property and restriction information, we achieve relatively high
matching qualities of 0.86 and 0.88 for tasks 248 and 249. The removal of properties in task 250
deteriorates F-measure to 0.67.

- 251 and 252: Here, the hierarchies have been altered partially by flattening and extension. We
observe that this decreases match quality more than the removal of the complete hierarchy in task
248.

- 253 to 257: Two out of hierarchy, instance, and property information have been removed from the
ontologies. For task 253 and 257 we achieve almost the same quality as in task 248 and task 250,
respectively. In task 254, only instance information is left, and only a small portion of mappings
can be found, resulting in a low F-measure of 0.43.

- 258 to 261: Again, hierarchies have been flattened or expanded. The results for tasks 258 and 259
are equal to those of tasks 251 and 252.

- 262 to 266: Here, all property and hierarchy information is removed, or altered. Additionally,
information contained in instances is removed. Still, instance identifiers are retained. Therefore,
the quality for those tasks is similar to that of tasks 254, 260, and 261, respectively. Obviously, if all
instance information was completely removed, we would not be able to find any correspondences
for task 262.
Figure 5.4 shows the best obtained qualities for series 3xx, which features four real-world ontologies developed at MIT, University of Maryland, Karlsruhe University, and INRIA. Due to their independent development, those tasks exhibit more irregular heterogeneities to the reference ontology than do the tasks of series 1xx and 2xx. For example, the hierarchies exhibit mismatches of various degrees, instead of being totally removed or systematically flattened. Additionally, none of the ontologies contains instance data. Despite the relatively high difficulty of the 3xx series, we reach a good average F-measure of 0.87, as shown in Figure 5.5.

- 301 to 304: The target ontology of task 304 is very similar to the reference ontology, sharing many similar class and property names. Consequently, we achieve almost optimal quality for task 304 (F-measure 0.96). For tasks 301 and 303, we observe relatively high similarities in the hierarchies and labels of entities. For example, a property labelled hasAuthor corresponds to a property author in BibTeX. Therefore, we achieve good quality for those tasks as well. The quality is lowest for task 302 (F-measure 0.77), which is very dissimilar to BibTeX, for example, in its hierarchical structure.

5.4.2 Generic Matching Evaluation

Above, we presented the best obtainable matching results we achieve when customizing the matching process to the different tasks. The test results presented in this section reflect the best matching quality obtained by applying a generic matcher configuration on all tasks, as described in Section 5.3.

The chart shown in Figure 5.6 indicates the F-measure value we achieve for each task by applying the Strategy matcher with the configuration details given in Section 5.3. Figure 5.7 shows the Precision, Recall, and F-measure values averaged over the four task series and all tasks. We observe a high total quality, with Precision at 0.96, Recall at 0.87, and F-measure at 0.91.

In addition to the experiments we carried out using Strategy, we also calculated the best average matcher result on basis of the $2^{10} - 1$ matcher combinations executed before for the task-specific evaluation, as described in Section 5.3.

- **Best average matcher results** The matcher that produces the best overall results averaged over the complete series is composed of 6 constituent matchers: Comment, Instance, RelationStat, Children, StatPropagation, and LexPropagation. When using this best average matcher, we achieve a result with a total Precision of 0.92, Recall of 0.86, and F-measure of 0.88. As expected, this result falls below the quality we obtain when using Strategy. Still, the best average matcher combination can solve most tasks at a very high level.
We have already commented in detail on the qualities of the customized matching results in Section 5.4.1. Therefore, we do not want to discuss the individual task results of Strategy, since the reasons that lead to these results are in many cases similar to the argumentation in the previous section.

Rather, we want to highlight the difference we observe when comparing the generic qualities to the best obtained match qualities presented in Section 5.4.1. We illustrate the deviation of match qualities in Figure 5.8. It is obvious that Strategy on average yields results of lower quality than the task specific matcher configurations. This is because Strategy is configured in a generic way, and therefore often contains constituent matchers which deteriorate the matching result to a certain degree. However, it is interesting to observe that for over half of the tasks, the generic Strategy matcher yields either the same or only slightly worse qualities than do the customized matchers. For those cases, the Strategy configuration is apparently a very good choice, exploiting available features optimally. Note that very small F-measure deviations can in most cases be regarded as negligible in this comparison, since the tasks contain only a small number of 29 to 97 intended mappings. Therefore, a decrease of F-measure by, for example, 0.02 corresponds to only one or two additional false positives or false negatives in the matching result. We now want to comment on such tasks where the quality produced by Strategy deviates to a larger degree (0.04 to 0.12) from the best obtainable quality.

Due to the complex coaction of the individual matcher results in the composite matcher, we can give no precise indication as to what exactly causes the observed deviations. However, analyzing the characteristics of the concerned tasks, we identify the following as likely causes for the lower matching qualities produced by Strategy:

- **Complex name heterogeneity:** For task 209 and tasks 301 to 303, we observe that the quality
obtained with the Strategy matcher falls below the best obtained quality by over 0.06 to almost 0.12, which is the highest deviation in the series. In those tasks, the heterogeneity in the ontologies is (partially) due to relatively complex naming modifications, that is, replacing names by their synonyms. In addition, the ontologies do not contain comment data or comments are highly heterogeneous. The complex matcher combination applied within Strategy seems not to be able to account well for those special mismatches.

- **Hierarchy modification:** We also observe that there is a correlation between lower generic task qualities and modifications in the hierarchy. This is specially manifest for the real-world tasks and tasks where ontologies were subject to hierarchy flattening and expansion, which cause diffuse modifications to the class hierarchy. By “diffuse”, we mean that part of the hierarchy is still preserved, leading to a relatively high overall structural similarity factor $\text{sim}_{\text{struct}}$. For this reason, hierarchical matchers are included in the Strategy matcher configuration, which can deteriorate the matching result for some tasks, especially so if no naming information is available in the matching process. Correspondingly, we see quality deviations of 3% to 4% for tasks 222, 230, 237, 260, and 265, where hierarchies have been flattened. The high quality loss in tasks 301 to 303 could also be partially due to their irregular hierarchical heterogeneity. For tasks where hierarchies are expanded or removed, such as tasks 238, 248, 252, 253, and 259 we observe negligible deviations of about 2%.

![Figure 5.8: Quality difference between customized matchers and Strategy](image.png)

5.5 Analysis and Assessment of Individual Approaches

Due to the wide range of individual matchers that are applied in our methodology, it is not possible to conduct a comprehensive evaluation as to how different combinations of matchers perform. Such an evaluation would require us to analyze a great number of matcher combinations (exponential in the number of applied constituent matchers). Therefore, we rather want to look at the results of some individual matchers in more detail. This should also provide a more intuitive insight to the reader.

Unfortunately, it was not possible to evaluate all of the implemented matchers, since the available matching tasks do not include all available modelling features of OWL DL. This concerns, for instance, matchers that leverage equivalence information or property characteristics. Another problem when examining the influence of individual matchers in the matching process is that many matchers do not deliver useful results by themselves. Rather, we in fact must combine them with other matchers in such a way that different features complement each other in the overall process. This is, for example, true for matchers that evaluate the structural or relational statistics of elements. Such matchers discover most intended mapping pairs based on their similar statistics, but at the same time also find many false positives. Applied individually, they therefore often yield a high Recall, but also a very low Precision, thus resulting in a low F-measure. In combination with other approaches, however, such matchers often add the “final
clue” to distinguish two otherwise equally similar element pairs. For the mentioned reasons it is difficult to assess the influence single matchers contribute to a final match result in a precise, objective way. In the remaining section we therefore at times resort to a subjective assessment of benefits and shortcomings of individual approaches, so as to provide the reader with an intuitive understanding.

### 5.5.1 Combining Constituent Matchers

First, we provide two examples to show how individual constituent matchers influence the matching results reported in Section 5.4.1. Particularly, the examples illustrate how the matching quality for tasks 209 and 303 develops when different matcher combinations are considered, starting with the matching results for single matchers, and then showing the combined results.

Figure 5.9: Matcher results for task 209

Figure 5.10: Matcher results for task 303

Figure 5.9 shows the matching qualities for the matchers that, when combined, yield the best matching result for task 209. Due to the altered entity names in this task, the best matcher combination contains only such matchers that leverage instance data, property information, and structural or relational statistics of elements. Note the different behaviour of matchers Instance and RelationStat. While Instance delivers mappings with maximum Precision, the Recall of this matcher is relatively low. This is due to the fact that only a part of the elements has associated instance data. RelationStat, on the other hand, has a very high Recall, but delivers also a high number of false correspondences, thus producing a very low Precision. This is because relation statistics can be derived for almost all elements in the ontologies, which leads to correct mappings but also to false ones, since many elements have similar relation statistics. The other two matchers StatPropagation and Property show a considerably better result, where both Precision and Recall are very high and yield an F-measure of 0.88 and 0.84, respectively. Since those two matchers are themselves relatively complex composite matchers, they already utilize many different features. This leads to a high Precision since only such correspondences where most constituent matchers agree are returned as mapping. Furthermore, StatPropagation applies iterative propagation, thereby adjusting similarities within the graph. When combining StatPropagation and Property, we yield an F-measure of 0.92. This falls below the best matching result (with all four matchers) by only about 0.01, which is negligible.
5.5. ANALYSIS AND ASSESSMENT OF INDIVIDUAL APPROACHES

In Figure 5.10 we can observe a similar result for the matchers that yield the best solution for task 303. Here, the applied matchers rely mostly on linguistic information and propagation, leveraging label, comment, and hierarchical information. The Comment matcher delivers relatively low Precision and Recall, because the available comment data\(^1\) is quite heterogeneous. The Name matcher delivers a very high matching quality of 0.76, as well as Children. The LexPropagation matcher produces slightly lower results, though still on a high level. When combining the two first matchers, we obtain a quality increase to about 0.80, which is due to the fact that they leverage different information — comment and label data — which complement each other and lead to an increase of Precision. Again, we see that the combination of all considered matchers does not yield substantial further quality improvement.

Indeed, we observe a similar behaviour for most matching tasks. That is, in most matcher combinations a single constituent matcher produces the best part of the matching quality, whereas complementing matchers only add slight improvements to the result.

5.5.2 Lexicon-based Approach

As described in Section 4.4.1, we implemented a basic lexical matcher aiming to deduce mappings to linguistically related words (e.g., synonyms, hypernyms) by finding corresponding relationships within the WordNet taxonomy. In the following, we show how such a matcher performs when being applied individually and in composition with a standard Name matcher. Again, we will not carry out a detailed evaluation, but present and comment on some results we obtained when testing the matcher.

Naturally, the lexicon-based approach can not bring any improvement of quality for most tasks of the series, since they do not contain synonym labelled elements. In the following, the matchers are applied to task 209, where many labels are replaced by synonyms, and to the real-world task 303. As in the previous section, we show the obtained qualities for a number of matcher combinations. Figures 5.11 and 5.12 show the results for task 209 and task 303, respectively. The applied matchers include Name, the lexicon-based matchers TaxRel(3) and TaxRel(2), where the maximum allowable path length in the WordNet taxonomy is 3 and 2, respectively, and Name TaxRel. Furthermore, we indicate the results that are obtained when running the previously described Strategy matcher applying lexicon-based matchers (Strategy(Lex)) or only simple Name matchers (Strategy(Name)).

The presented results show clearly that the lexicon-based matcher in its present form cannot improve matching results significantly. Indeed, being applied individually, it yields qualities that are below those of simple Name matcher application, as can be seen when comparing the figures for Name, TaxRel(3), and TaxRel(2) in the charts. Only when composing the Name and TaxRel matchers in the Name TaxRel matcher, we obtain a slight improvement in Recall and F-measure for task 209. When applying Strategy(Name) to tasks 209 and 303, we obtain a matching quality that is about 0.04 lower then that for Strategy(Lex). Averaged over the complete task series, the improvement in the overall quality is hardly perceivable, thus not justifying the high performance costs the lexicon-based approach causes.

The lexicon-based matchers identify some correct synonym correspondences, thus producing a number of correct mappings, which would not have been discovered by normal name matching. The drawback of the approach is, however, that it also finds many false positives due to relatively short paths in the WordNet taxonomy that suggest a relation between actually unrelated words. Some returned false correspondences are obviously incorrect, while others might be feasible in a different context. For example, the matcher returns a mapping between properties country and state, which could be considered correct in certain application contexts.

Observing the results described in this section, we see that we still need to invest much effort in order to achieve a better utilization of lexical resources in the matching process. For example, the parameteri-

---

\(^1\)The Karlsruhe ontology used in task 303 does not provide comment annotations, but existing label annotations are used instead.
5. Evaluation of the Prototype

5.6 Comparison of Matching Qualities

Above, we presented the results of two experiments we conducted to evaluate COMA++(O) with respect to its customizability and generic applicability. In this section, we compare the obtained results to those of other systems. For this comparison we consider four systems. First, we investigate the quality improvement of COMA++(O) against the previous version of the system, COMA++. Then, we compare COMA++(O) against three state-of-the-art systems, which participated in last year’s OAEI evaluation campaign: RiMOM [YLT06], Falcon-AO [HCZ+06], and another version of COMA++ [MER06], which was enhanced at Leipzig University, and is herein after denoted as COMA++(LE).

For a sound comparison of the evaluation results, we must distinguish between our two evaluation hypotheses and corresponding experimental settings. First, we compare the results of the customized matchers, as presented in Section 5.4.1. Then, we compare the generically obtained results from Section 5.4.2.

5.6.1 Quality of Customized Matching

The results we achieved when processing each of the tasks with a customized matcher cannot generally be compared against the OAEI contest results, since the experiment setting does not comply with the contest guidelines. Therefore, we first compare those results against the schema matching framework COMA++. Unfortunately, this does not provide an account of how well the methodology performs in comparison to the general state of the art. However, it can give us an insight as to how the utilization of semantic features and auxiliary information influences the matching result for different tasks.

A second comparison shall be carried out against the results of RiMOM, one of the participants in the
5.6. COMPARISON OF MATCHING QUALITIES

OAEI contest [YLT06]. As described in Section 2.7.2, RiMOM is a generic matching algorithm and allows no task-specific configuration of the matching process. Therefore, the result that the generic process produces for a specific task can to a certain aspect be viewed as the best result that can be achieved by RiMOM for that task. This said, we regard the results reported in [YLT06] as detached from the guidelines of the contest and use them for comparison with the results from our customized matching process. At the same time, we acknowledge fully that this comparison is somewhat hampered by the fact that the two systems follow different approaches, with the RiMOM system being completely focused towards a general applicability that does not allow any customization.

Comparison to the Underlying Implementation COMA++

We compare the best customized matching results as reported in Section 5.4.1 against the best qualities we obtain using COMA++ on the test series. The quality difference is illustrated in Figure 5.13. To obtain the results for COMA++, we used the same procedure as reported in [Do05]. That is, analogue to our experiment setting, we chose for each task the best combination out of $2^8$ combinations of matchers implemented in COMA++. We hold that those results can be used as a basis for comparison, even though they might be further improved by customized parameterization of the matching process.

![Figure 5.13: Comparison of the best matching results of COMA++(O) and COMA++](image)

For most of the tasks, COMA++(O) yields a substantial improvement in matching quality. This is especially true for such tasks where naming information is discarded completely, such as tasks 201, 202, and all tasks of series 2xxb. For those tasks, the (F-measure) quality advances by 0.16 (task 201) to 0.57 (tasks 250 and 258). We also observe improvements for the tasks which exhibit considerable heterogeneity of names, such as tasks 205 – 207, 209, and 210. Here, the improvement in matching quality is 0.05 to 0.15. The high difference in the matching qualities can be ascribed mainly to three factors:

- **Utilization of instance information:** Although already equipped with instance matching functionality, COMA++ does not extract instance information from ontologies. In COMA++(O) we provide for according parsing functionality and are thus able to leverage the instance matching approach of COMA++. Since in the provided test series, instance information is equal for all ontologies (except where it has been discarded), it contributes important information to the description of an element. In fact, we can in a way view it as another sort of label information, which naturally contributes valuable information to the matching process. When ignoring instance data, the quality advance for most tasks is still high, about 50% of the values reported in Figure 5.13.

- **Distinction of class and property semantics:** The distinction of class and property semantics in the retrieval of related elements and the conduct of applied matchers plays an important role for obtaining the high qualities reported in Section 5.4.1. Conversely, the lack of such distinction in COMA++ hampers mapping discovery heavily where no naming information is provided, as
can be seen when looking at the results for the tasks of series 2xxb. Additionally, the fact that COMA++(O) only allows mappings between ontology elements of the same type also improves the matching results. Indeed, by making this assumption, simple name matching is sufficient in all such tasks where naming information is remained intact. COMA++ on the other hand needs additional information even in those cases, since it cannot distinguish between a property and a class with similar names.

- **Utilization of restriction semantics:** In contrast to COMA++, which ignores restriction information completely, COMA++(O) introduces this information in the matching process both for the similarity computation of classes and properties. Since restriction information is available to a high degree in all but a few tasks of the benchmark, it amounts to an important factor in the element descriptions.

Concluding, the comparison of COMA++(O) to the generic ontology matching approach of COMA++ shows a substantial advance in matching qualities. Averaged over the tasks of the different series, we obtain a total increase in F-measure values of 0.04, 0.42, 0.04 for series 2xxa, 2xxb, and 3xx, respectively. This amounts to a total F-measure increase of 0.14 over all tasks of the series.

**Comparison to RiMOM** Figure 5.13 shows the difference between the F-measure values achieved by RiMOM in the OAEI contest 2006 and those obtained by the customized matching process of COMA++(O).

As expected, the differences in quality we obtain when comparing our methodology to RiMOM are less distinct than those observed for COMA++. The first observation is that COMA++(O) yields equal or slightly better quality results in such tasks where naming and comment information is retained or modified to a certain degree, that is, in all tasks of series 1xx and all but one of series 2xxa. In contrast to that, the matching qualities for many tasks in series 2xxb and 3xx diverge considerably, either exceeding or falling below the quality of RiMOM by up to almost 0.15 or 0.05, respectively. Unfortunately, as yet there is no publication describing RiMOM in detail, which inhibits a precise evaluation as to the causes of the observed quality differences between the systems. As reported in Section 5.5, it is also a non-trivial task to determine the influence individual COMA++(O) matchers have on a composite matching result. Still, we can conjecture what might be the cause for the observed deviations:

- **Generic vs. task-specific configuration:** The first, and very obvious, difference is that RiMOM was run in a generic configuration, whereas COMA++(O) used task-specific matchers.

We highlight that this of course contributes a lot to the higher qualities we obtain. We assume that

---

2 We derived the corresponding F-measure values from the reported Recall and Precision values in [YLT06].
this is the main reason for the slightly higher quality in the tasks of series 2xxa, particularly in tasks 208 to 210, where the task-specific configuration of COMA++(O) allows a better adaption to the specific linguistic heterogeneity.

- **Fix-point propagation vs. constrained propagation:** The fix-point propagation approach applied by RiMOM seems to be superior to the simple constrained propagation we apply in COMA++(O). In RiMOM, propagation is conducted iteratively until all semantic information is fully leveraged. In COMA++(O), on the other hand, we only do one or a fixed number of propagation steps. However, there are also tasks in series 2xxb where COMA++(O) outperforms RiMOM considerably, such as task 251, 252, 258 to 261, and 266, as well as all tasks of series 3xx. We assume this advance might be owed to the restricted, “diffuse” hierarchic heterogeneity due to flattening and expansion of hierarchies in those tasks. Such heterogeneity might affect the propagation mechanism applied within RiMOM negatively.

Averaged over the tasks of the different series, the quality provided by the customized matchers exceeds that of RiMOM by 0.03, 0.05, and 0.01 for series 2xxb, 3xx, and over all tasks, respectively.

### 5.6.2 Quality of Generic Matching

When using a generic matcher as described in Section 5.3, we comply with the OAEI contest settings. Therefore, we can compare the results obtained using the *Strategy* matcher against the results of participants in the OAEI 2006 campaign. We choose for this comparison the RiMOM [YLT06] system, which performed best in the OAEI contest 2006, Falcon-AO [HCZ06], and COMA++(LE) [MER06].

In Figure 5.15 we present the average quality (F-measure) that each of the systems achieved over the different series. To facilitate comparison, we indicate in Figure 5.16 the quality difference between the results achieved by each of the three systems and those achieved by the generic matcher *Strategy*.

![Figure 5.15: Generic quality of different prototypes averaged over the series](image)

We report that the *Strategy* matcher of COMA++(O) yields matching results comparable to those of current state-of-the-art systems. Particularly, we achieve qualities that are on the same level as those of the highest-ranking contest participants in the OAEI 2006 evaluation campaign. In the following, we briefly discuss the observed deviations with respect to the different systems.

**RiMOM** In general, we observe almost similar overall qualities for COMA++(O) and RiMOM, with only slight deviations over the series. Both systems achieve an overall F-measure of 0.91 over all tasks. While RiMOM outperforms COMA++(O) in series 2xxa and 3xx, COMA++(O) achieves a better quality in the tasks of series 2xxb. Except for series 3xx, the results correspond roughly with the observations made in Section 5.6.1, where we assumed the different propagation schemes of the tools as a cause for the quality deviations. In fact, when looking at the detailed qualities of single tasks, we make the
same observations with respect to flattening and removal of hierarchies as described in Section 5.6.1. In particular, we see that COMA++(O) outperforms RiMOM by up to 0.14 in tasks where hierarchies have been flattened or expanded, while it falls below RiMOM by up to 0.10 in those tasks where either hierarchy or property information was removed completely.

**Falcon-AO** COMA++(O) outperforms Falcon-AO in series 2xxa and 2xxb by about 0.02 and 0.09. For series 3xx, COMA++(O) falls about 0.02 below the results of Falcon-AO for the same tasks. We observe that the quality deviations for the tasks in series 2xxa are mostly negligible, except for such tasks where naming information has been removed or altered considerably. In those tasks, the quality achieved by COMA++(O) exceeds that of Falcon-AO by up to 0.13. In series 2xxb, COMA++(O) achieves a much higher quality in several tasks, exceeding the quality of Falcon-AO by up to 0.20 (task 250 and 257). We observe that the highest deviations are in tasks where property information has been removed, which leads to the conjecture that COMA++(O) can handle such heterogeneity better than the structural matching component GMO of Falcon-AO. Over all tasks, Falcon-AO achieves an F-measure of 0.88, which is below the quality of COMA++(O) by about 0.03.

**COMA++(LE)** Our system exceeds the F-measure values achieved by COMA++(LE) by 0.01, 0.09, and 0.05 in series 2xxa, 2xxb, and 3xx respectively. Over all tasks, the F-measure of COMA++(LE) falls below that of COMA++(O) by 0.03. In general, we make the same observations as for COMA++ in Section 5.6.1 with regard to the distinction of class and property semantics and the utilization of restriction information. COMA++(O) achieves much higher qualities than COMA++(LE) in most tasks of series 2xxb and 3xx, and task 202, where name information is removed or heterogeneous to the reference ontology. Here, the distinction of class and property semantics in COMA++(O) proves superior to the realization in COMA++(LE), where no such distinction is made. Additionally, COMA++(O) produces considerably higher matching quality for tasks 228, 233, 236 and 239 – 247, where property information is removed. This confirms the above argument: Since COMA++(LE) treats properties equivalent to subclasses, the removal of such properties is likely to deteriorate the quality of its hierarchy-based matchers.

### 5.7 Assessment of the Proposed Methodology and Implementation

Concluding the evaluation of the customized and generic matching process of COMA++(O), we observe a considerable improvement in matching quality compared to the (schema-based) generic ontology matching approach of COMA++. In addition to very good task-specific results, our system produces also
good generic matching results over all tasks, competing at a high level with the best participants of last year’s OAEI evaluation campaign. In particular, we achieve almost the same quality as the highest-rating contest participant RiMOM. At the same time, we can leverage the flexible configuration facilities of COMA++, extended by additional integration functionality as reported in Section 3.5. This enables us to customize the matching process of COMA++(O), thereby further improving match quality, as reported in Section 5.6.1.

We consider the following aspects decisive for the good quality of our methodology:

- **High utilization of ontology semantics:** The utilization of the broad range of semantics in OWL DL ontologies benefits the discovery of mappings, especially in such cases where no naming information is available and only hierarchic and/or relation-based information can be leveraged. Since we can exploit a wide variety of features in the matching process, we are also able to apply the composite matching approach in a more modular, and thus more effective fashion. That is, we can specify more fine-grained matchers leveraging the different features in an optimal manner. This helps to account for different mismatch types in matching tasks and therefore fosters the customizability of the process. For example, by distinguishing class and property semantics we can address more selectively the mismatches related to the hierarchy or to properties in a specific task. Apart from that, the broad range of utilized features also helps the genericity of the process. This becomes evident when we observe the high quality of matching results we obtained using the best average matcher combination, as briefly reported in Section 5.4.2.

- **Separation of element kinds:** In the similarity computation process, we distinguish by default the class and property elements of an ontology. Thereby, we restrict the potential mappings to element pairs of the same type. This enables COMA++(O) to exclude many element pairs from the matching process at an early stage. This way we can, for example, achieve very good matching qualities even with only basic name matching in tasks where names are homogeneous.

- **Automatic match process configuration for generic matching** The automatic matcher configuration helps to create a high-quality generic matching process based on the composite matching approach, as reported in Section 5.4.2. This approach relies on the provision of a broad range of matchers that can be used in the configuration.

Restrictions of the methodology Apart from the described advances we observe in the matching quality of COMA++(O), we also want to report important issues that can restrict the applicability and effectiveness of the proposed methodology, depending on the requirements of the application context:

- **High sensitivity in the configuration of matchers:** One restriction we observed in the evaluation of the system is its high sensitivity to minor adjustments in the matcher configuration. This affects above all the composition of the different matcher sets in the configuration of the StrategyMatcher.
  Moreover, we became aware of the high importance of providing matchers that distinguish well between different kinds of feature information, such as label-, hierarchy- or relation-based information. Only by applying such matchers that consider different features of ontology elements in a well-separated way can we account effectively for different kinds of mismatches.
  A user who configures the matching process must therefore have some insight in the specifics of the different approaches. This restricts to a certain extend the applicability of the methodology.

- **Restricted mappings:** Currently, we restrict the possible mappings by default to mappings between entities of the same type. Thereby, we prohibit mappings between properties and classes. However, such mappings might be feasible and desired in other mapping tasks than those used in the evaluation.
  Furthermore, the proposed solution is restricted to the discovery of mappings that express equality (or, more general, semantic similarity) between two elements. We have not yet investigated ways
5.8 Benefits and Shortcomings of the Conducted Evaluation

Apart from the above considerations, we also identify some critical issues that arise due to the nature of the conducted evaluation. This section addresses both positive and negative aspects of the evaluation, which are important for an objective assessment of the presented results.

Ensuring comparability of evaluation results  For the evaluation we used the publicly available benchmark series of the OAEI 2006 evaluation contest. Although we could not participate in the OAEI 2007 contest, we tested our methodology in compliance with the published contest guidelines. That is, as Section 5.3 highlights, for evaluating the generic applicability of COMA++(O) we used no manual configuration. Furthermore, we applied no auxiliary data other than the publicly available lexical database WordNet. By complying with the contest guidelines, we can compare our generic evaluation results against those of current state-of-the-art systems that participated in the previous contests. This allowed us to investigate how we performed on the different tasks compared to other systems, and to identify thereby the strengths and weaknesses of the proposed methodology.

Evaluating task influence on matching quality  The different matching tasks contained in the benchmark series are in most cases versions of the BibTex reference ontology, where different features, such as class labels or the concept hierarchy, have been modified or discarded. Thus, each of the tasks exhibits a different sort of ontology mismatch (or combination of several such mismatches), depending on the modified features. The test series is therefore a good basis for evaluating strengths and weaknesses of different matching approaches, that is, matchers exploiting different features of ontology elements, with respect to different tasks.

Toy data  The most obvious weakness of the conducted evaluation is that the tasks consist essentially of relatively simple ontologies with a low number of elements, which furthermore have not been developed independently of each other, but are generated from the reference ontology (except for series 3xx). Furthermore, the modifications on label, structure, and property information are always carried out on either none or the whole set of the concerned elements (hierarchy flattening and expansion might be considered a slight exception, since only a few classes are affected by the modification). While this facilitates the development and evaluation of single matching approaches as pointed out above, it does not fully account for the heterogeneity in real-world matching tasks.

For the stated reasons, the evaluation results can not be regarded as a true measure for the actual performance of the system, but rather — as is the intention of the contest — as a basis for comparison with other tools. Other matching tasks published in the context of the OAEI campaign require matching between large real-world directory structures and thesauri. Those tasks are, however, not suitable to develop or investigate the quality of our methodology with respect to the utilization of ontology semantics.

Tuning of the methodology  Due to the lack of further appropriate test data, we used the benchmark series both in the development and the evaluation of our methodology. As in other fields such as machine learning and information retrieval, this approach implies the risk that the methodology and the single matchers are tuned towards the series. Different test bases would be necessary in order to properly separate the development and evaluation of the methodology.


**Modelling features used in the benchmark ontologies**  One design goal in the development of the proposed methodology was to optimally utilize the broad range of semantic features in OWL DL ontologies in the matching process. The OAEI benchmark series allowed us to test many matchers and evaluate their influence in the ontology matching process. This includes the hierarchy-based matchers adopted from COMA++, the matchers that exploit relation and restriction information, the PropagationMatcher strategy, and, to a certain extent, the lexicon-based matcher. Unfortunately, we could not test the influence of some other matchers, because the features they utilize are not applied in the task ontologies. This includes, for example, the matchers that utilize equivalence and disjointness information.

**5.9 Summary**

We carried out a comprehensive evaluation of COMA++(O) to investigate the matching quality provided by the proposed methodology. The prototype was applied on the benchmark series of the OAEI 2006 evaluation campaign. We deployed two experiment settings aiming at evaluating both the customizability and generic applicability of our methodology. For testing the generic applicability, we complied with the guidelines of the OAEI evaluation campaign so as to ensure comparability to the contest participants.

The tests yielded high-quality results for both settings. Applying task-specific matcher configurations, we obtain an overall F-measure of 0.92 over all tests of the series. This amounts to a considerable improvement in quality compared to the quality obtained by COMA++. The generic matching configuration of COMA++(O) also yields good quality results throughout the test series, competing at a high level with the contest participants of the OAEI 2006 campaign. In particular, we obtain a very good F-measure of 0.91 over all tasks, which is comparable to the result of last year’s best contest participant RiMOM.

The systematic test series provided a good base for investigating the influence of different aspects of the proposed methodology. In particular, comparing the results for different tasks and systems, we observe that the fine-grained utilization of semantic features, above all the semantic distinction of classes and properties, benefits the matching process and improves matching quality.

We observe that the test cases of the benchmark series consist of relatively small-sized ontologies and that the different mismatches in the tasks do not fully account for the heterogeneity that can occur in real-world matching tasks. Due to the lack of test data and the constrained time frame, we could not perform a more comprehensive evaluation of our methodology. In future, it would be desirable to carry out tests on different realistic ontologies, preferably from the business domain, to further evaluate the real-world applicability of COMA++(O).
5. EVALUATION OF THE PROTOTYPE
6 Conclusion

This chapter summarizes the presented work and discusses possible improvements of the proposed methodology as suggestions for future research.

6.1 Summary

During the last years, the increasing use of ontologies in distributed environments has created the need for effective mechanisms to enable interoperability between heterogeneous ontologies. The task of ontology matching plays a central role in all ontology reconciliation problems. This thesis addressed the development of a methodology for semi-automatic matching of ontologies.

First, we performed an extensive evaluation of the current state of the art in ontology matching. Therein, we discussed how various kinds of heterogeneity between ontologies necessitate different approaches to solve the matching problem, and presented the wide range of techniques currently applied in the field of ontology matching.

In the development of our methodology, we placed emphasis on two aspects we identified as decisive for an effective approach to the ontology matching problem. First, our methodology should optimally exploit the spectrum of semantic features in ontologies. Second, we aimed to realize a flexible matching process that is both highly customizable and generically applicable to a diverse range of matching tasks. To achieve those goals, we designed the methodology as a framework providing different matching approaches to be integrated in a complex matching process. We introduced the concept of element descriptions, which represent the semantics of ontology elements as the entirety of their features in the ontology. Then, we described how matchers can compute the similarity of ontology elements based on their description, and how different matchers can be integrated flexibly in order to utilize the available information optimally.

Furthermore, we provided functionality for automatic configuration of the matching process so as to enable optimal matching quality when no manual configuration is possible.

We implemented the proposed methodology based on the generic framework architecture of the schema matching tool COMA++. For the prototype COMA++(O), we extended the basic functionality of COMA++ in order to enable the utilization of semantic features expressed in OWL DL ontologies. We further implemented a library of ontology matchers and strategies that can be used to integrate those matchers.

To investigate the quality of our ontology matching methodology, we evaluated COMA++(O) using a series of ontology matching tasks published as a benchmark for the OAEI evaluation contest. We performed a number of experiments on the benchmark using different matcher configurations. In doing so, we tested both the customizability and generic quality of our methodology. For both criteria, the results of the evaluation are promising.

In particular, we achieved a considerable advance in matching quality compared to the basic ontology matching functionality in COMA++. Furthermore, the generic matching process yielded high-quality results comparable to those of other state-of-the-art tools for ontology matching. We observe that the utilization of the semantic features in ontologies, as well as the high flexibility of the proposed methodology, were key factors for achieving the reported results.
6.2 Directions for Future Work

For further improvement on the presented methodology and implementation, we identify various possible directions for future research, which we review below. Apart from investigating such extensions to the functionality of COMA++(O) we aim to further evaluate the methodology on different matching tasks and benchmarks, and to apply the tool to a realistic scenario within a current research project.

**Extending language support** In Section 2.4 we briefly investigated the issue of language-level mismatches and discussed two common ontology representation languages in that context. Currently, COMA++(O) supports the matching of ontologies described in OWL DL. For a wider applicability of the tool it would be desirable to support more ontology languages, such as F-Logic. However, as we have highlighted in Section 2.4, different ontology languages vary in their expressivity and often exhibit subtle differences in their semantics. Therefore, enabling additional language support and accounting for language-level mismatches is an involved issue that must be based on a comprehensive study of the respective language.

**Investigating automatic configuration** The effort and expertise a user has to invest to configure the matching process should be minimized as far as possible. Therefore, further automation of the matching process will be an important step to improve the real-world applicability of ontology matching tools. As indicated in Section 3.6, there are much more approaches to automatic configuration than the basic approach used in our methodology. Particularly, beyond selecting matchers that shall be applied in the matching process, an automatic configuration approach could, for example, aim at assigning suitable weights for individual matchers, or at optimally parameterizing basic similarity measures for a given task.

**Visualization of ontologies and mappings** Although automation of the matching process is one of the central issues in ontology matching research, it is generally acknowledged that there will always remain a certain necessity for a user to check and adapt the obtained results. To this end, the matching task and resulting mappings should be visualized in a clear and intuitive form. COMA++(O) adapts the COMA++ visualization, displaying ontologies in the form of a simple tree, where different semantic information is distinguished only through conventions in the labelling of nodes. Visualizing ontologies in a more intuitive way could enable a user to better grasp the semantics of the visualized data. For example, a graph structure using differently styled and coloured nodes and edges can be applied. A multitude of ontology representations have been devised (see, for example, [AEC+06]). However, it is still an open research issue how such visualization can be used in the special application task of ontology matching. A tool that addresses the specific needs of ontology editing and maintaining tasks is, for example, the Protégé plug-in Jambalaya [LS05]. The OLA [EV04] system implements a graph-based visualization of ontologies and the mappings between them, which, however, seems adequate only for very small ontology matching tasks. The central issue that needs to be addressed in this context is how the layout of ontology graphs should be performed in order to place adequate focus on the relevant elements and their mappings. For example, when a user selects a particular element or mapping, this could cause highlighting of different parts of the ontologies, zooming in to elements that are mapped, or nesting nodes based on different criteria.
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>B2B</td>
<td>Business-To-Business</td>
</tr>
<tr>
<td>BPM</td>
<td>Business Process Management</td>
</tr>
<tr>
<td>BPMO</td>
<td>Business Process Modelling Ontology</td>
</tr>
<tr>
<td>C-OWL</td>
<td>Context OWL</td>
</tr>
<tr>
<td>COMA</td>
<td>COMbining MAtchers</td>
</tr>
<tr>
<td>CWA</td>
<td>Closed World Assumption</td>
</tr>
<tr>
<td>DL</td>
<td>Description Logics</td>
</tr>
<tr>
<td>DOLCE</td>
<td>Descriptive Ontology for Linguistic and Cognitive Engineering</td>
</tr>
<tr>
<td>ESSI</td>
<td>European Semantic Systems Initiative</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>JWNL</td>
<td>Java WordNet Library</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>OAEI</td>
<td>Ontology Alignment Evaluation Initiative</td>
</tr>
<tr>
<td>OLA</td>
<td>OWL Lite Alignment</td>
</tr>
<tr>
<td>OWA</td>
<td>Open World Assumption</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PLM</td>
<td>Product Lifecycle Management</td>
</tr>
<tr>
<td>PMO</td>
<td>Process Mining Ontology</td>
</tr>
<tr>
<td>PSL</td>
<td>Process Specification Language</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RiMOM</td>
<td>Risk Minimization based Ontology Mapping</td>
</tr>
<tr>
<td>SBPM</td>
<td>Semantic Business Process Management</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SUMO</td>
<td>Suggested Upper Merge Ontology</td>
</tr>
<tr>
<td>SUPER</td>
<td>Semantics Utilized for Process Management</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modelling Ontology</td>
</tr>
</tbody>
</table>
Bibliography


List of Figures

1.1 Example matching task .................................................. 6
1.2 The matching process .................................................. 11

2.1 Local matching between ontologies ................................. 16
2.2 Matching to a global common ontology .......................... 16
2.3 Cluster-based matching ................................................. 16
2.4 Integration of matching approaches ............................... 25
2.5 Categorization of individual matcher approaches .............. 27

3.1 Conceptual overview of the matching methodology ............ 38
3.2 Description features example ......................................... 41
3.3 Typed directed graph structure ....................................... 46
3.4 Example graph representation ........................................ 47
3.5 The composite matching process ...................................... 55
3.6 Dependent matcher execution ......................................... 57
3.7 Automatic match process configuration ........................... 59

4.1 COMA++ system architecture ......................................... 64
4.2 The matching process in COMA++ ................................. 65
4.3 Implementation of the ontology graph structure .................. 68
4.4 Creating a new DependentMatcher ................................. 78
4.5 Creating a new StrategyMatcher ...................................... 79
4.6 Creating a new PropagationMatcher ............................... 79

5.1 Best qualities for series 1xx ........................................... 86
5.2 Best qualities for series 2xxa ......................................... 87
5.3 Best qualities for series 2xxb ......................................... 88
5.4 Best qualities for series 3xx ........................................... 88
5.5 Best qualities averaged over the series ............................ 89
5.6 Strategy task qualities ................................................ 90
5.7 Strategy task qualities averaged over the series ............... 90
5.8 Comparison of customized matcher application and Strategy matcher application . 91
5.9 Matcher results for task 209 ........................................... 92
5.10 Matcher results for task 303 .......................................... 92
5.11 Application of name and lexicon-based matchers to task 209 .. 94
5.12 Application of name and lexicon-based matchers to task 303 .. 94
5.13 Comparison of the best matching results of COMA++(O) and COMA++ .. 95
5.14 Comparison of the best matching results of COMA++(O) and RiMOM .. 96
5.15 Generic quality of different prototypes averaged over the series .. 97
5.16 Quality difference to other systems averaged over the series .. 98
List of Tables

2.1 Categorization of input data ................................................................. 23
3.1 Description features of ontology elements ............................................. 42
3.2 Ontology graph representation .............................................................. 46
3.3 Basic similarity measures ................................................................. 49
4.1 Hybrid matchers ........................................................................... 73
4.2 Composite matchers ........................................................................ 75
4.3 DependentMatcher configuration example ............................................ 77
4.4 StrategyMatcher configuration example .............................................. 77
4.5 PropagationMatcher configuration example .......................................... 78
5.1 Characteristics of the OAEI benchmark tasks ......................................... 84