Instance-Based Ontology Matching and the Evaluation of Matching Systems

Inaugural-Dissertation

zur

Erlangung des Doktorgrades der

Mathematisch-Naturwissenschaftlichen Fakultät

der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Katrin Simone Zaiß

aus Düsseldorf

November 2010

Aus dem Institut für Informatik
der Heinrich-Heine Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Referent: Prof. Dr. Stefan Conrad
Koreferent: Prof. Dr. Martin Lercher
Tag der mündlichen Prüfung: 10.12.2010

The Answer to the Ultimate Question of Life, the Universe, and Everything:

101010

(loosely based on Douglas Adams' novel The Hitchhiker's Guide to the Galaxy)

ACKNOWLEDGEMENTS

This thesis is the result of three and a half year of research at the Databases and Information Systems Group of the Department of Computer Science at the Heinrich Heine University of Düsseldorf.

First of all, I acknowledge my advisor and first referee *Prof. Dr. Stefan Conrad* for supporting me in my research and for creating a motivating and comfortable working atmosphere. I am very thankful for the possibility to work under his supervision and to learn from his experiences. I also thank the second reviewer of this thesis, *Prof. Dr. Martin Lercher*, for his interest in my work and willingness to be the second referee.

My special compliments go to my colleagues and friends, *Sadet Alčić* and *Tim Schlüter*, who helped me with their technical knowledge and their patience, and who enlightened each working day with their great sense of humor. It was a pleasure to share the office with Sadet, and I thank Tim for the joint work. Additionally, I extend my compliments to my new colleagues, *Ludmila Himmelspach*, *Jiwu Zhao* and *Thomas Scholz*, and to my former colleague, *Johanna Vompras*, who supported me with her experience.

Special thanks to *Guido Königstein*, *Marga Potthoff*, and *Sabine Freese* for their guidance in technical and administrative issues.

My deepest and warmest thanks go to my family, my parents *Sabine and Kurt Zaiß*, my brother *Max* and my sister *Alexa.* They support me in each situation and without their love and their mental support, this thesis would not have been possible.

Last but not least, I deeply want to thank *André Teloo* for supporting me each and every day with his patience, his understanding and his ability to always look on the bright side of life.

Düsseldorf, Germany
November, 2010

Katrin Simone Zaiß

Abstract

The matching of heterogeneous information sources is a crucial task in many different domains. In order to find relations between the different pieces of information, which are annotated using different structures and formats, matching systems have been developed. In the past two decades, ontologies became more and more important as a way to represent the semantics of information in a machine read- and processable way. Hence, many ontology matching systems have been developed as well, which make use of the different parts of ontologies to resolve the heterogeneities. Most systems focus on the exploit of schema or structure information, but ontologies also provide instances, which express the semantics of a concept independent of its meta information. Current instance-based matching methods give room for improvements in several aspects. Matching Systems also need to be evaluated using appropriate test data. Existing benchmarks are not sufficient for testing instance-based methods. In this thesis, we focus on the development of instance-based matching methods, their combination with schema- and structure-based methods and their evaluation.

We introduce two novel instance-based matching methods. The first method makes use of regular expressions or sample values to characterize the concepts of an ontology by their instance sets. The second approach uses the instance sets to calculate many different features like average length or the set of frequent values. Both approaches finally compare the characterizations, i.e. the regular expressions or the features, to obtain similarities between the entity sets of two (or more) ontologies. An alignment between the ontologies is then obtained by examining the similarity set.

In order to test single matching methods or complex matching systems well-defined test benchmarks have to be available, preferably including the correct alignments to facilitate the evaluation. Current benchmarks do not enable extensive studies on instance-based methods, because the number of instances is significantly too low. We present an additional benchmark, ONTOBI, which can be used to test instance-based methods, but also all other kinds of matching algorithms or systems.

Finally, we present `MICU`, a complex matching system which unifies the advantages of instance-, schema- and structure-based matching methods combined with an efficient user feedback interaction. In order to speed up the process alignments of previous matching cycles are reused.

Zusammenfassung

Heterogene Informationsquellen findet man in vielen unterschiedlichen Gebieten und das Matching (der Abgleich) dieser Quellen ist ein Prozess, der häufig gebraucht wird. Um die Verbindungen zwischen den verschiedenen Informationen, die unterschiedlich formuliert und struktiert sein können, zu finden, wurden Matching-Systeme entwickelt. Als Struktur zur maschinenles- und verarbeitbaren Repräsentation von Wissen wurden in den letzten zwei Jahrzehnten Ontologien immer populärer. Folglich wurden auch viele Ontologie-Matching-Systeme entwickelt, welche die unterschiedlichen Elemente der Ontologien untersuchen um die Heterogenitäten zwischen den Ontologien aufzulösen. Dabei verwenden die meisten Systeme hauptsächlich Schema- und Strukturinformationen, obwohl Ontologien auch Instanzen enthalten, welche die Bedeutung der Konzepte unabhängig von jeglichen Meta-Informationen beschreiben. Die bisher existierenden instanzbasierten Methoden bieten noch einigen Raum für Verbesserungen. Diese Arbeit beschäftigt sich mit der Entwicklung neuer instanzbasierter Methoden, ihrer Kombination mit schema- und strukturbasierten Methoden und ihrer Evaluation.

Zu Anfang werden zwei neue instanzbasierte Methoden vorgestellt. Der erste Ansatz verwendet reguläre Ausdrücke oder Beispielwerte um Konzepte einer Ontologie mit Hilfe ihrer jeweiligen Instanzmengen zu charakterisieren. Die zweite Methode berechnet aus der Instanzmenge verschiedene Features (Merkmale) wie die Durchschnittslänge oder die Menge der am häufigsten vorkommenden Werte. In beiden Fällen werden die Charakteristika, d.h. die regulären Ausdrücke oder die Feature-Werte, verglichen um eine Ähnlichkeit zwischen den verschiedenen Elementen der zwei Ontologien zu berechnen. Die paarweisen Ähnlichkeiten werden dann verwendet um die Korrespondenzen zwischen den Ontologien zu finden.

Um einzelne Methoden oder komplexe Matching-Systeme testen zu können, braucht man geeignete Testdaten-Sets, in denen idealerweise auch direkt die Menge der Referenz-Korrespondenzen enthalten sein sollte. Bisher verfügbare Benchmarks bieten jedoch nicht die Möglichkeit instanzbasierte Methoden ausführlich zu testen, da nicht genügend Instanzen vorhanden sind. Mit ONTOBI präsentieren wir einen zusätzlichen Benchmark, mit dem man instanzbasierte, aber auch alle anderen Arten von Matching-Methoden oder -Systemen, testen kann.

Abschließend stellen wir mit `MICU` ein komplexes Matching-System vor, welches die Vorzüge von instanzbasierten mit denen von schema- und strukturbasierten Methoden kombiniert und dabei effizient mit dem Benutzer zusammenarbeitet. Zusätzlich werden die Ergebnisse früherer Matching-Durchläufe wiederverwendet.

Contents

1 Introduction

1.1 Motivation

Knowledge is an important resource of humankind. The development of the Internet facilitates the distribution and the accessibility of all kinds of information. The information concerning one domain of interest can be structured and displayed in many different ways, i.e. it is presented heterogeneously. In most cases a human can cope with this heterogeneity, but for computers it is quite difficult to capture the semantics of the information. To enable the "understanding" of knowledge for computational applications, ontologies have been developed. Originally, ontologies have been used in the context of philosophy and information theory. "An ontology is an explicit specification of a conceptualization", this definition of an ontology has been introduced by Thomas R. Gruber [Gru93]. Thus, an ontology describes a specific domain of interest by including concepts and the relations among them. A hierarchical taxonomy as it is used to describe biological relationships between races can be represented by an ontology. The goal of an ontology in computer science is that it adds semantics to the data such that an application can classify the information.

Example 1. *Imagine having a website of an institute of computer science. This website may include various names, images and email addresses. Using the structure and the graphical representation of the website, a human can link the names to the images and find the correct email address of a person. For computer applications it is rather difficult to connect these pieces of information, especially because the information might be represented heterogeneously on different websites. Ontologies can be used to connect the information that belongs to one person. An ontology describing the website might include a concept that is named "Person". Each person has a name, a surname, a* *title, an image and an email address; these properties are called attributes. For each person that is displayed on the website, an instance of the concept "Person" is created, and the values of the attributes are stored and linked to the person as well. A computer application is now able to search for a person and find the correct email address without using any heuristics or considering the graphical representation.*

So, how is the information stored? Ontologies need to be represented in a structured way, such that they can be easily processed by applications. Since XML has been developed for this purpose, it can be used to represent ontologies as well. But semantics can not be expressed with XML. Hence, languages like RDF or OWL have been developed, which use the syntax of XML and extend it with the possibility to add semantics. The information is always stored in triples consisting of subject, predicate and object. The used predicates are often predefined such that there is a common understanding of the meaning. This structured way of representing the knowledge enables the extraction of the desired information for adequate applications. Although this reduces the structural or syntactical heterogeneity of the knowledge, there is still heterogeneity that is caused by humans. In the example described above the designer of the ontology could name his concept "Employee" instead of "Person" and the attributes could be named in an other way, too. Additionally, the division of the domain of interest into concepts and attributes might differ depending on the developer or the framework in which the domain is embedded. This kind of heterogeneity cannot easily be detected or solved automatically by any application extracting the information.

Example 2. *After working at the Technical University of Munich, a professor currently works at the University of Duesseldorf. There is information available about this professor on the websites of both universities. Imagine a person who searches for information about this professor. Using a normal search engine, both websites might appear in the results and the user has to examine them to find the current working place and the correct email address. If ontologies are used, the application (e.g. a semantic search engine) might directly find the recent email address because both websites contain concepts that belong to the professor, and the one existing on the Munich website includes an attribute like "discarded on" with a date in the past. But this only works if the application is able to detect the similarity of both concepts using the concept definition. In the case that the concepts are described heterogeneously the application cannot find a connection. To solve the heterogeneity problem a matching system can be used.*

Matching systems have been developed to find correspondences between all kinds of information sources (i.e. files, databases, ontologies etc.). They represent a kind of artificial intelligence that tries to rebuild the human interaction with computer systems and the knowledge of the relations between the different concepts in the world. In the case of ontology matching systems similar concepts have to be found as automatically as possible. According to the different types of heterogeneity, different kinds of strategies have been developed. We mainly focus on ontology matching approaches, but most of the strategies can be adapted to all kinds of information resources. In general, the strategies can be divided into schema-, structure- and instance-based ones. Schema-based matching methods aim to find similar concepts by regarding the names of concepts and attributes independent of the used structure and/or expression, e.g. misspellings, homonyms and synonyms should be detected. Structure-based matching methods use the relations between the entities and the graphical topology of an ontology to detect correspondences. Some ontologies also contain instances, i.e. concrete values of concepts. These instances provide a huge amount of information and are also used for matching purposes. The instance sets are compared element-wise or a feature set is calculated which provides the basis for a comparison.

The matching of ontologies is needed in many other applications as well. An overview will be given in the next subsection.

1.2 Application Areas

The variety of application scenarios in which ontology matching is a crucial tasks is very high. [ES07] define six central use cases for ontology matching:

- ontology engineering
- information integration
- peer-to-peer information sharing
- web service composition
- autonomous communication systems
- navigation and query answering on the web

Ontology engineering describes the process of designing, editing or versioning ontologies. If a new ontology should be created, there might be the wish or the need to reuse ontologies or to combine existing ontologies. To support the designer during this process, a matching system can search for correspondences. This is especially important if the ontologies are very big. In some cases, there might be different versions of an ontology, because it has evolved and the previous versions still exist. If a system uses the old version and wants to upgrade to the new one, a matching system can help to find differences between the versions.

The most important application scenario for matching systems is *information integration.* Whenever multiple heterogeneous information sources shall be used together for a common task (e.g. query answering), the sources have to be matched and integrated (physically or at least virtually). A matching system produces the connections between the sources by finding similar terms, concepts or attributes; these connections provide the basis for the integration process. Imagine having two companies that aim to cooperate or merge. As a consequence, their data (e.g. employee data) has to be integrated to a common data source. In most cases the particular databases or information sources are structured heterogeneously using different terms etc and so on, such that a matching algorithm could facilitate finding the correct correspondences.

In *peer-to-peer networks*, systems may exchange data. If the peers are totally autonomous (as they should be), they might describe their data using different kinds of concepts or labels. To enable a reasonable information exchange anyway, the different descriptions (e.g. ontologies) have to be matched.

Web services may present their interfaces using different languages. A data matching process is needed to match the service descriptions as well as the inputs and outputs.

Autonomous communication systems like agents often have to exchange messages. The outer form of the messages is determined by an agent communication language, but the content of the message is expressed according to a local ontology. To understand the message, an agent has to match his ontology to the ontology of the sender.

The *answering of queries* is an important issue for the web. Search engines are used very often and they work with ontologies to refine queries, too. Each question, that is posed by a user, is translated into terms of the local ontology; this is done by a matching process. Meta search engines translate a query into the terms of the different ontologies, that are provided by the underlying search engines, collect the results and translate them back according to the original ontology.

1.3 Contributions of this Thesis

The matching of ontologies is important for many different tasks. In the last decades a lot of research has been done in the field of information source matching. In most cases the matching strategies focus on the use of schema or structure information. There are also some approaches that use the instance sets to find correspondences between the concepts, but there is still room for improvements. The goal of this thesis is to develop new instance-based matching methods, to enhance existing ones and to combine them with appropriate schema- and structure-based methods. As a result a matching system should be obtained, which interacts with the user who can prove and correct the matching results.
In detail, the contributions of this thesis are:

- The main contribution of this thesis is the presentation of two novel instance-based matching methods. The first one makes use of regular expression and catchword lists to characterize the instance sets. Using these lists, a vector representation is created for each instance set. These vectors provide the basis for calculating similarities and finally for deriving a matching.
  The second approach uses the instance set to calculate a set of well-defined features that describe the instance set. These features are collected in a vector and compared by using different similarity measures. The obtained attribute-to-attribute similarity is propagated upwards to finally match concepts, too.

- Another contribution of this work is the development of an evaluation framework for testing ontology matching systems. This framework, ONTOBI, is helpful for all kinds of matching systems, but has also been developed with regard to instance-based matching methods. Existing evaluation benchmarks mainly focus on schema- and structure-based methods, because the number of instances is very small (if there are some actually). ONTOBI is an evaluation benchmark, which contains a huge set of different matching tasks; the ontologies consist of many classes and the data set contains more than 13000 instances. For each matching task, a reference alignment is given such that the evaluation is facilitated.

- For creating ONTOBI, an ontology modificator has been developed which gets an ontology as input and applies several modifications such as addition/deletion of comments, extraction of synonyms, insertion of spelling mistakes or flattening of the hierarchy. The modifications can be chosen manually and the resulting ontology is created in the same format as the input ontology. Additionally, a reference alignment can be produced, which describes the relation between input and output ontology. Using the ontology modificator one can produce its own evaluation benchmark.

- A further contribution is the presentation of a new matching system, which combines the new instance-based matchers with well known schema- and structure-based matchers. This matching system also includes a user feedback mechanism which enhances the matching quality. Additionally, it provides the possibility to evaluate the matching results by comparing the produced alignment with a reference alignment.

1.4 Outline of this Work

This thesis is organized as follows: Chapter 2 contains a general introduction and gives an overview of the theoretical background of this thesis. First, ontologies are briefly introduced including the description of the different elements as expressed in the ontology languages RDF, RDFS and OWL. Then, the field of matching heterogeneous information sources is introduced by giving a problem definition and representing the different kinds of matching methods. Finally, the different aspects that are important for evaluating matching systems are presented.

In Chapter 3 we give an overview of the state of art by introducing several matching systems and evaluation frameworks.

Chapter 4 includes the presentation of the major objective of this thesis, i.e. the development of novel instance-based matching methods. First, an approach using regular expressions is described, followed by a method, that uses features to find mapping correspondences.

These methods are included in a more complex matching system which is described in Chapter 5.

Chapter 6 gives an overview of the benchmark developed for the evaluation of matching systems and especially for instance-based matching methods.

The proposed matchers and the matching system are evaluated using different test scenarios; the results are presented and discussed in Chapter 7.

The thesis concludes in Chapter 8.

2 Background

The matching of heterogeneous information sources is a problem which occurs in many application areas; the information sources might be database schemas, XML schemas, ontologies or other models. Since this thesis focuses on the matching of ontologies, we reduce the matching problem to an ontology matching problem. In the following, a few foundational aspects are clarified. First of all, a short introduction to the Semantic Web is given in Section 2.1 and the term "ontology", as it is used in the context of this thesis, is explained in detail in Section 2.2. The different languages that can be used to model an ontology are presented in Section 2.3. A definition of the matching problem itself is given in Section 2.4. After that, similarity measures, which provide the basis for any matching algorithm, are presented in Section 2.5. Section 2.6 gives an overview of the different kinds of matching systems and algorithms. In Section 2.7 the principles of the evaluation of matching systems are explained in detail. A short summary in Section 2.8 concludes this chapter.

2.1 The Semantic Web

The World Wide Web is a collection of billions of documents that store a huge amount of data and information. In most cases one and the same information is stored in many different, heterogeneously structured documents. Although search engines have been improved over the last years, it is sometimes still difficult for a user to find the best answer to his question in an appropriate time. In the Semantic Web [BL],[BLHL01], an idea of Tim Berners-Lee, the director of the W3C (World Wide Web Consortium), documents are annotated with additional meta data, such that software agents can interpret the information to capture the semantics. This is not only limited to the use within the WWW but can also be used within several applications. The most simple way to add meta data to information is the use of RDF (see Section 2.3), which describes relations between different pieces of information. Meta information can also be used not only to better describe information sources but to better understand the questions of the user. Hence, search queries can be adapted using a specific knowledge base which e.g. "knows" that business and trade denote the same. An interesting development is the search engine, or as the inventors say "knowledge engine" [wol10], *Wolfram—Alpha* which is able to answer questions directly without only displaying websites containing the search words. Asking "Which day of the week was 23rd of August 1983?" the user gets the answer "Tuesday" and additional information about the time that has past since then or special events on this date are displayed.

2.2 Ontologies

Ontologies are a relatively new way of defining and storing knowledge. In the context of the Semantic Web, ontologies represent a way to add meta information to the contents included in a website, such that software agents can understand the meaning of the information and better work with it.
In general, an ontology describes a certain domain by dividing it into several concepts and describing the *relations* between those concepts. A *concept* (also called *class*) is the biggest component (concerning the information content) of an ontology. It may be described by several *attributes*, which are concrete data fields (data type properties, see Subsection 2.3). It is also possible to store information directly as *instances* (also named data values) of concepts. In addition, most ontologies provide extra information on the entities, e.g. data types or comments. An exemplary ontology is displayed in Figure 2.1 and the different parts of the ontology are named in Example 3:

Example 3. *The ontology in Figure 2.1 displays a few pieces of information of the domain "organization of a university" stored in an ontology.*
Professor, address, chair and lectures are concepts or classes. The pale blue fields like name or surname are called attributes or data type properties. The diamonds represent relations between different concepts; a relation might be an object property or (not shown in this example) a subclass property. Instances are concrete values of the attributes and are displayed in italic strings. An instance belonging to the concept 'professor" is Stefan Conrad, where Stefan is the value of the attribute "name" and Conrad the value of "surname". This instance has a relation to the instance conrad@cs.uni-duesseldorf.de.

Summarized an ontology contains the following elements or entities: concept, attribute, relation and instance.

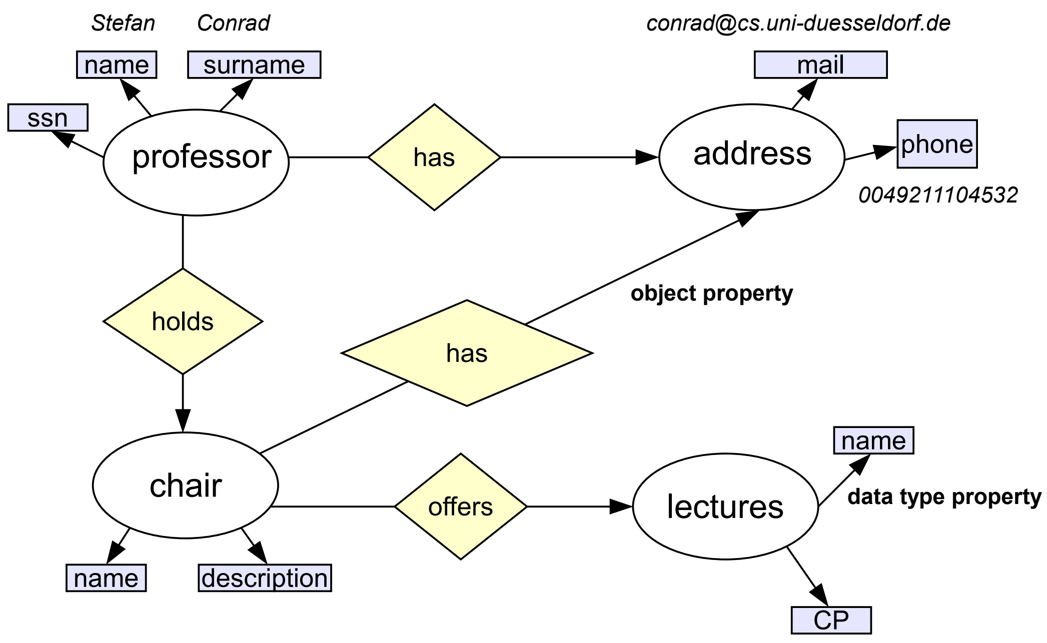


Figure 2.1: Example Ontology O1

To describe the meaning of the terms more exactly we will give a formal notation in the following. An ontology is a tuple $O =< C, A, I, R, D >$ such that:

$C = \{c_1, ..., c_k\}$ is the set of concepts.
$A = \{A(c1), ..., A(ck)\}$ with $A(cl) = \{a_{l1}, ..., a_{ln}\}$ being a set of attributes assigned to a concept c_l.
$R = \{r_1, ..., r_m\}$ with $r_p \in C \times C \times \sigma$ is the set of relations; a relation connects two concepts with each other, σ denotes the natural alphabet in which the name/type of the relation is expressed.
$I = \{I_1, ..., I_k\}$ with $I_n = \{i_1, ..., i_o\}$ being a set of instances assigned to a concept c_n.

Concept instances can be divided into several attribute instances. An attribute instance denotes the value of a concrete attribute within a concept instance.
After introducing the different languages, that can be used to express ontologies, in the next section, these entities will be connected to specific elements of the language.

2.3 Ontology Languages

In the Semantic Web software agents should be able to understand the content of web pages. Ontologies are one possibility of representing knowledge and correlations between different pieces of information. To be automatically processable for software agents, the knowledge has to be expressed as simple as possible in an adequate format. For this purpose several languages have been developed which will be explained in the following.

2.3.1 Resource Description Format

RDF (Resource Description Framework) [Bec04] is a language used for describing resources and is recommend by the W3C. RDF uses a simple data model which is understandable for humans and machines, the *RDF Model*. The information is presented as a statement, a so-called *triple*, that always consists of three parts: *subject*, *predicate* and *object*. The predicate represents a binary relation between the subject and the object.
An example for a statement is: Stefan Conrad has the family name Conrad.
In this case *Stefan Conrad* is the subject, *has the family name* is the predicate and *Conrad* is the object. As the names states, RDF is used to describe resources (in the Web), so the entities are connected to *URIs* (Uniform Resource Identifiers, see [BLG05]). Additionally, the use of URIs ensures the unambiguous identification of the resources that are used. Subjects and predicates are usually resources (expressed in URIs), whereas an object may be a resource or a literal. *Literals* are Unicode strings representing a concrete value. Plain literals are literals that have an appended language tag, typed literals are extended by a data type URI. Whenever a resource is needed to organize a group of other resources or to represent an unnamed relation, *blank nodes* can be used as subjects or predicates.

The RDF statements may be represented as directed graphs, the so called RDF graphs. The graph displaying the example described above is shown in Figure 2.2. Resources are represented as ovals, arcs display the predicates, where the arc always points from the subject to the object, and rectangles symbolize literals.

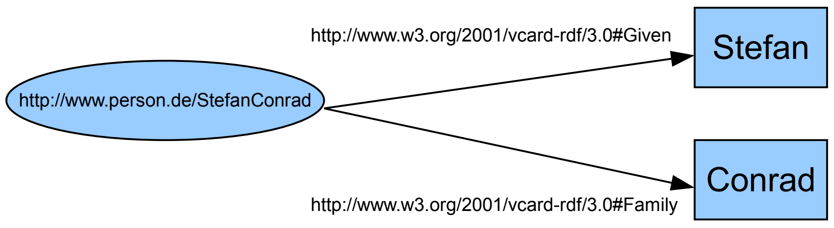

http://www.w3.org/2001/vcard-rdf/3.0#Given
Stefan
http://www.person.de/StefanConrad
Conrad
http://www.w3.org/2001/vcard-rdf/3.0#Family

Figure 2.2: A sample RDF graph

RDF statements can be represented using different kinds of *syntaxes*. As mentioned before, RDF statements are often expressed in a XML syntax, also called *RDF/XML*. Another possibility is the N3 (Notation 3) syntax [n306], which is better readable for humans and nicely displays the triple structures of the statements. In the following, both languages are used to express the RDF graph of Figure 2.2.

The following text shows the translation of the RDF graph into RDF/XML.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#">
  <rdf:Description rdf:about="http://www.person.de/StefanConrad">
    <vcard:Given>Stefan</vcard:Given>
    <vcard:Family>Conrad</vcard:Family>
  </rdf:Description>
</rdf:RDF>
```

First of all, the used XML version has to be stated. The second tag includes the definition of namespaces; `xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"` sets the variable `rdf` to the URI mentioned consecutively. This variable can be used in the following document instead of the whole URI which improves the readability and saves storage space. `rdf:RDF` is the root element of each RDF document. Using the `rdf:about` attribute, the resources, i.e. the subject that is described in the following within the `rdf:Description` tag, is introduced. The following tags `vcard:Given` and `vcard:Familiy` are the properties (predicates) of the resource stated with `rdf:about`. The data values, i.e. the literals, are included within the property tags.
In this example, the objects are always literals. If the resource of our example `Stefan Conrad` has a photo available in the web, which is a resource with a URI, the corresponding RDF/XML document describing this fact looks like this:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#">
 <rdf:Description rdf:about="http://www.person.de/StefanConrad">
  <vcard:photo rdf:resource="http://dbs.cs.uni-duesseldorf.de/conrad.jpg" />
 </rdf:Description>
</rdf:RDF>
```

The resource denoting the image, i.e. the object in this case, is directly included in the property tag `vcard:photo` and a `rdf:resource` is prefixed.

As mentioned before, N3 is another syntax for representing RDF graphs. N3 is better readable for humans and offers the possibility to express logics (e.g. by allowing the use of variables).

Our example in N3:

```
@prefix : <http://www.w3.org/2001/vcard-rdf/3.0#> .
<http://www.person.de/StefanConrad>     :Family "Conrad";
        :Given "Stefan" .
```

Similar to the RDF/XML syntax it is possible to define namespaces in N3 by using `@prefix`. Each statement ends with an "." and it is possible to concatenate different properties describing the same resource by an ";".

2.3.2 Resource Description Framework Schema

RDF is only usable for describing resource and relations between them; it is not possible to create new properties and classes or to create a taxonomy. For this reason, an extension of RDF, *RDF Schema* (RDFS) [BG04], has been developed which allows the definition of classes. A class denotes a group of resources; in other words, a resource is an instance of a class. RDFS extends RDF especially with the following classes and properties:

- Classes (self-explaining)
  - `Class`
  - `Resource`
  - `Literal`
  - `Datatype`
  - `Property`
- Properties
  - `range`: states that the values of a property are instances of the specified class
  - `domain`: states that any resource that has the according property is an instance of the class
  - `type`: states that a resource is an instance of the specified class
  - `subclassOf`: states that all instances of one class are also instances of another class
  - `subPropertyOf`: states that all resources related by one property are also related by another
  - `label`: represents a human-readable label of the resource
  - `comment`: represents a human-readable description of the resource

The RDF Model can be used without changes, i.e. each RDFS statement is also an RDF graph and the RDF/XML syntax can also be used. The following example shows the usage of the `Class` concept:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.person.de/">
<rdfs:Class rdf:ID="Person" />
<rdfs:Class rdf:ID="Professor">
  <rdfs:subClassOf rdf:resource="Person"/>
</rdfs:Class>
</rdf:RDF>
```

A class Person is introduced using `<rdfs:Class rdf:ID="Person" />`.
The subject of this statement is `http://www.person.de/Person`, the object is `http://www.w3.org/2000/01/rdf-schema#Class` and the, not obvious, property is `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.

2.3.3 Web Ontology Language

The Web Ontology Language (OWL) [SCW04] is another extension of RDFS and provides additional vocabulary to describe more semantics, e.g. relations between classes, characteristics of properties etc. It is the standard language for defining ontologies. There are three sublanguages of OWL:

- OWL Lite: simplest version; enables the creation of taxonomies and simple axioms
- OWL DL: extends OWL Lite with the possibility to express full description logics, but with some restrictions to ensure computational completeness and decidability, e.g. a class cannot be an instance of an class
- OWL Full: like OWL DL but without restrictions

In the following, the most important elements of OWL, that we will need for further explanations, are explained.
A class is the most basic element of an ontology and its definition is made using the following statement:

```
<owl:Class rdf:about="http://www.person.de/Professor"/>
```

Individuals, also called instances, are defined as instances of certain classes using the following statement:

```
<Professor rdf:ID="http://www.person.de/StefanConrad"/>
```

This is the simplest way of assigning an instance to a class, which equals the more complex statement (with additional specification of the property `vcard:Given`):

```
<owl:Thing rdf:ID="http://www.person.de/StefanConrad" />
<owl:Thing rdf:about="http://www.person.de/StefanConrad">
   <rdf:type rdf:resource="Professor"/>
   <vcard:Given>Stefan</vcard:Given>
</owl:Thing>
```

In this example, the property `vcard:given` has the value "Stefan" for this determined instance; this is what we call attribute instance. In general, properties are binary relations and using OWL, we can distinguish between two types of properties: datatype properties and object properties. Datatype properties define a relation between classes and literals (together with the assignment of XML datatypes). Using the properties `rdfs:domain` and `rdfs:range` the datatype property can be restricted. A statement creating a datatype property called `name`, which belongs to the domain of `Professor`, i.e. it is an attribute of `Professor`, and which contains string values, is the following:

```
<owl:DatatypeProperty rdf:about="http://www.w3.org/2001/vcard-rdf/3.0#name">
    <rdfs:domain rdf:resource="http://www.person.de/Professor"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Object properties define a relation between two classes. The properties `rdfs:range` and `rdfs:domain` are also defined for this type of property. The following example statement defines an object property belonging to the class `Professor` which is named `has` and which connects `Professor` to the class `Address`:

```
<owl:ObjectProperty rdf:about="http://www.person.de/has">
    <rdfs:range rdf:resource="http://www.person.de/Address"/>
    <rdfs:domain rdf:resource="http://www.person.de/Professor"/>
</owl:ObjectProperty>
```

According to the definition of an ontology given in Section 2.2, we finally want to connect the entities to OWL elements:

- a concept is defined as a `Class`
- an attribute is a `owl:DatatypeProperty`
- a relation is either a `owl:ObjectProperty` or a `rdfs:subclassOf`
- a (concept) instance is assigned to a class using the `rdf:type` property
- an attribute instance is the literal that is related to a concept instance by a specified attribute

2.4 The Matching Problem

An ontology can be used in many different application areas to describe and store knowledge. In general, there are many different ontologies describing the same domain. In most cases, the ontologies are not totally equal, because the used vocabulary differs and the coverage of the domain is varying. If two ore more ontologies need to be compared, matched or integrated, correspondences have to be found despite the heterogeneity. To demonstrate the different types of heterogeneities, Figure 2.3 shows an ontology which describes the same domain as the ontology displayed in Figure 2.1, but uses different terms.

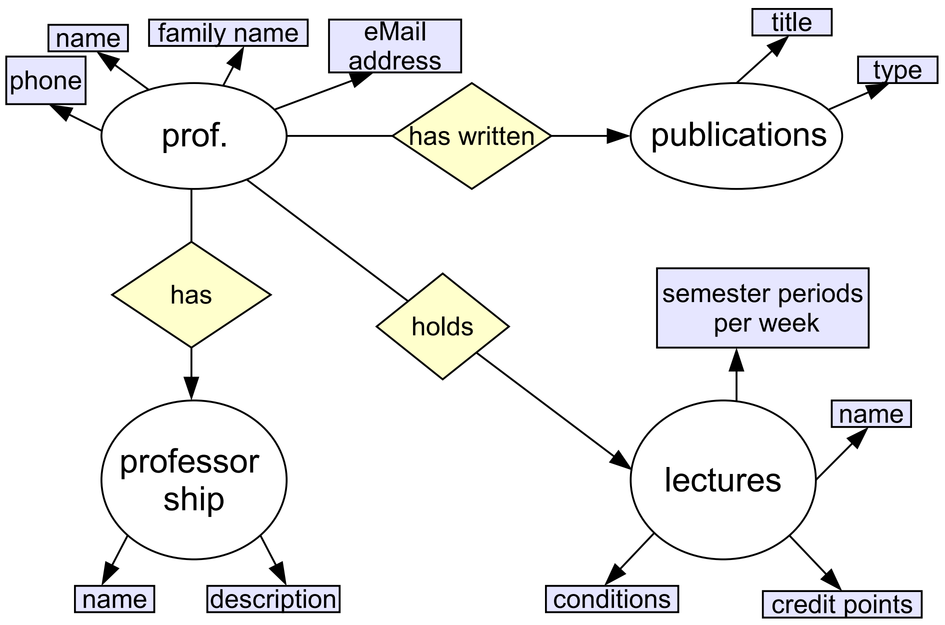


Figure 2.3: Example Ontology O2

[ES07] differentiates the following types of heterogeneity: Syntactic, terminological, conceptual and semiotic heterogeneity.

Syntactic heterogeneity is caused by the different representation formats of an ontology or knowledge base. RDF and OWL are only two possible languages, there are also formalisms like F-logic [KLW95]. Generally, this problem can be solved by finding equivalences between the different languages, preferably without loss of information. This kind of heterogeneity is not considered by most of the systems and it is not addressed in this thesis as well.

Terminological heterogeneity denotes the usage of different terms for the same (or a similar) real world thing. The labels of an entity might be expressed in different languages or synonyms are used, e.g. "surname" and "family name" in our example ontologies. Another example for this kind of heterogeneity is the use of abbreviations, like "professor" and "prof." in our example. Furthermore, the used terms can be different due to the specific context they

are used in. This kind of heterogeneity is difficult to cope with, because natural language evolves and the vocabulary differs among the people even if they speak the same language.
Conceptual heterogeneity (or semantic heterogeneity) is the collective term for several different possibilities to model a domain. It can be divided into three subproblems: difference in coverage, granularity and perspective.
Coverage differences occur, if ontologies are written from the same point of view, i.e. in the same context and with comparable vocabulary, but the part of the domain that is described differs and there are only overlapping parts. In our example the second ontology includes an additional concept "publications" which is not described by the first ontology.
Difference in granularity means that the same section of the domain is described but the depth of details is not equal. The concept "lecture" is included in both example ontologies but in the second ontology there are more attributes describing it.
If the point of view from which an ontology is designed differs, there is a *difference in perspective.*
Semiotic heterogeneity is caused by the subjective interpretation of the used terms by humans. Terms can be considered regarding their context or not, such that terms with the same semantics are interpreted in different ways. The first example ontology includes the attribute "CP" which stands for credit points. This is obvious when regarding the surroundings of this attribute, but the abbreviation itself can represent many different things.

A matching process aims to solve the different types of heterogeneities preferably automatically; a technically representation is shown in Figure 2.4, which has been published in [ES07]. A matching process can be represented as a function that matches two input ontologies o and o' by using a previous alignment A, a set of parameters and several other resources. As a result an alignment A' is produced, which represents the correspondences between the two input ontologies. Additionally, the kind of correspondence (equivalence, subclass etc.) and the confidence (similarity value) is given.

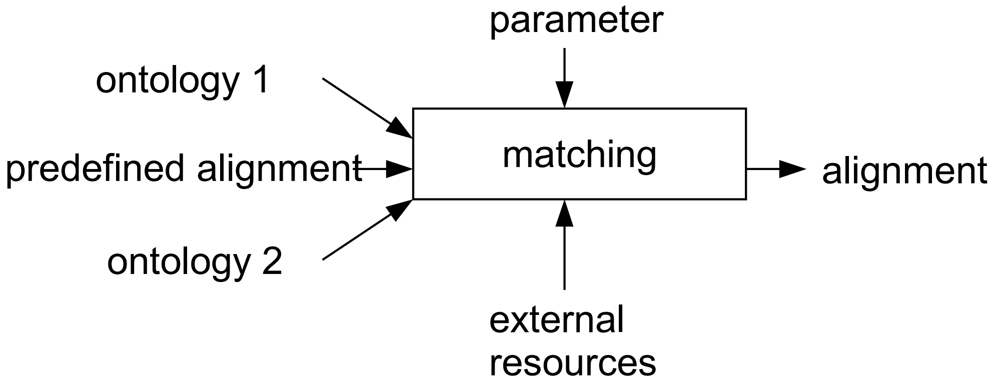


Figure 2.4: Matching Process, taken from [ES07]

The resulting alignment is defined as a set of correspondences, which represent relations between different entities. A correspondence can be described by a 5-tuple (see [ES07]):

$$< id, e, e', r, n >, \tag{2.1}$$

where id is a unique identifier of the correspondence, e is an entity of ontology o, e' an entity of o', r denotes an alignment relation, e.g. $=$ or $<$ and n gives a confidence value, e.g. a similarity value.
Depending on the matching algorithm the entities that can be combined within one correspondence can be limited, e.g. such that concepts can only be matched with concepts and not with attributes.
The notation given above describes so-called 1:1 (one-to-one) correspondences, in which exactly two entities take part. Due to difference in granularity or coverage in some cases there might also be 1:n (one-to-many) or even n:m (many-to-many) correspondences. Regarding our example ontologies the concepts `professor` and `address` of O1 match `prof.` in O2. For this purpose, the equation given above can be extended such that a correspondence can include more than two entities.

2.5 Similarity Functions

Similarity measures form the basis of all matching algorithms, because they determine whether two entities are similar or not. In the following, a formal notation of a similarity function as it used in the scope of this thesis is given:
A similarity function $sim : E \times E \to \mathbb{R}$ with $E = E_1 \cup E_2$, E_1 is the entity set of ontology $O1$, E_2 the entity set of O_2, which gets two entities as input and calculates a similarity value, has to satisfy some properties (see [ES07]):

- positiveness: $\forall x \in E, y \in E, sim(x, y) \geq 0$
- maximality: $\forall x \in E, \forall y, z \in E, sim(x, x) \geq sim(y, z)$
- symmetry: $\forall x, y \in E, sim(x, y) = sim(y, x)$

In this case, entities might be strings (e.g. concepts labels or comments), numbers (feature values) or sets of numbers or strings.
In the scope of our thesis we focus on similarity functions that calculate a (normalized) similarity value that is in the range of $[0, 1]$.
In some cases it might be useful to calculate distance values instead of similarity values. Distance functions fulfill the properties of similarity measures (but with minimality instead of maximality, i.e. $\geq$ has to be replaced by $\leq$).

Additionally, the following two properties hold for a distance function $dist : E \times E \rightarrow \mathbb{R}$:

- definiteness: $\forall x, y \in E, dist(x, y) = 0$ if and only if $x = y$
- triangular inequality: $\forall x, y, z \in E, dist(x, y) + dist(y, z) \geq dist(x, z)$

In the following we want to present a few of the most common similarity functions used in matching algorithms. We distinguish between string-based measures, numeric measures and set-based measures (mainly based on [ES07]). Another overview of several similarity measures can be found in [MYC08]; several distance measures are described in [WM97].
String-based measures:

The simplest possibility to determine the similarity of two strings is to test on equality. A corresponding measure can be defined as

$$sim_{equ}(s,t) = \begin{cases} 0 \text{ if } s = t \\ 1 \text{ if } s \neq t \end{cases}, \tag{2.2}$$

with s and t being two strings.
Strings might include spelling mistakes which results in a string equality of 0. For this purpose the n-gram measures can be used. An n-gram is a substring of length n, and n-gram(s, n) is defined as the set of all substrings of length n that can be obtained of string s. The n-gram measure for two strings s and t is defined as follows:

$$sim_{ngram}(s,t) = n\text{-gram}(s,n) \cap n\text{-gram}(t,n) \tag{2.3}$$

This measure calculates a value $\in \mathbb{N}_0$; to normalize it to the interval $[0, 1]$, the following equation can be used:

$$sim_{ngramNorm}(s,t) = \frac{n\text{-gram}(s,n) \cap n\text{-gram}(t,n)}{min(|s|,|t|) - n + 1} \tag{2.4}$$

For short strings, the n-gram measure calculates relatively small similarities. e.g. in the case that only two letters are swapped. An example: the strings “label” and “laebl” do only have one 2-gram in common, which is “la”; consequently, the normalized 2-gram similarity of these two strings is $\frac{1}{4}$. To cope with this problem, small strings can be extended by adding some extra chars at the beginning and the end of the string (equally for both strings, that are compared).

The *edit distance*, is one of the most used dissimilarity measures. It calculates the number of operations that is needed to transform one string into another (and vice-versa). The allowed operations are not limited in general, but in most cases they consist of insertion, deletion and substitution. The cost which is assigned to each operation does not have to be equal but in most cases it is 1 (the Levenshtein distance, see [ES07]). Most easily the edit distance *ed* between two string s and t with $|s| = m$ and $|t| = n$ can be defined recursively:

$$ed_{i,j} = min \begin{cases} ed_{i-1,j-1} & +0 \text{ if } s_i = t_j \\ ed_{i-1,j-1} & +1 \text{ (substitution)} \\ ed_{i,j-1} & +1 \text{ (insertion)} \\ ed_{i-1,j} & +1 \text{ (deletion)} \end{cases}, 1 \leq i \leq m, 1 \leq j \leq n \tag{2.5}$$

with $ed_{i,0} = i$ for $1 \leq i \leq m$ and $ed_{0,j} = j$ for $1 \leq j \leq n$.
In general, there are a lot of possibilities to normalize the edit distance to the interval $[0, 1]$ and to transform it into a similarity measure (if desired). The *String Similarity* is a measure based on the edit distance ed and calculates a similarity value between 0 and 1.

$$sim_{strsim}(c, d) := \max\left(0, \frac{\min(|c|, |d|) - ed(c, d)}{\min(|c|, |d|)}\right) \tag{2.6}$$

Number measures:

In some cases it might be demanded to compare numbers during the ontology matching process. Especially when comparing features like average length created using the instance set, a similarity of numeric values has to be calculated. One possibility is:

$$sim_{num}(a, b) = \frac{1}{1 + |(a - b)|}, \tag{2.7}$$

with a and b being two numbers.
Set-based measures:

In some cases it is useful not only to compare single values (strings or numbers), but also sets of them. One very popular measure is the *cosine similarity*, which calculates the similarity of two vectors expressed as the cosine of the angle between them. It is only applicable for vectors containing numbers and it is defined as:

$$sim_{cos}(v, w) = \frac{v * w}{|v| \cdot |w|}, \tag{2.8}$$

with v and w being two vectors, $|v| = |w|$. For ontology matching purposes, the cosine similarity is always in $[0, 1]$, because the values included in the vectors are always positive, i.e. we do not know of any approach that includes negative values within such vectors.

Another possibility to compare vectors is the *Minkowski* distance. As the name states, it calculates a distance, not a similarity, and is defined as

$$sim_{Mink}(v, w) = \left(\sum_{i=1}^{n} |v_i - w_i|^p\right)^{1/p}, \tag{2.9}$$

with $|v| = |w| = n$.
For $p = 1$ this distance is also known as the *Manhattan distance*, for $p = 2$ it is called the *Euclidean distance* and for $\lim_{p\to\infty}$ it is known as the maximums distance or the *Chebyshev*

distance.

The set-based similarity measures or distances presented so far can only be applied on vectors containing numbers. To compare sets of strings other similarity measures are needed. The simplest possibility is again to test on equality.

$$sim_{setEqu}(S,T) = \frac{|S \cap T|}{|S| + |T|}, \tag{2.10}$$

with S and T being two sets of strings.
In most cases it might be useful not to test on equality but on similarity. First of all, the similarity of all possible string pairs (s,t) with $s \in S$ and $t \in T$ is calculated and all pairs with a similarity above a certain threshold are counted. Formally, we calculate a set V with:

$$V = \{(s,t) | s \in S, t \in T, \text{ with } sim(s,t) \geq t, t \text{ is threshold}\} \tag{2.11}$$

The similarity of the two string sets is the calculated as:

$$sim_{setSim}(S,T) = \frac{|V|}{|S| \cdot |T|} \tag{2.12}$$

Similarity Aggregation:

In most cases, two or more similarity values are aggregated to a single one. The simplest possibility of combining x similarity values is given by:

$$sim_{agg}(sim_1, ..., sim_x) = \frac{\sum_{i=1}^{x} we_i \cdot sim_i}{\sum_{i=1}^{x} we_i}, \tag{2.13}$$

with we_i being weights assigned to the similarity values sim_i.

2.6 Classification of Matching Systems

Different surveys examining matching approaches can be found in the literature. The basis is established by [RB01], which has been extended in [SE05].

Matching algorithms may use different types of data as input, which provide a basis for dividing matching techniques. [SE05] propose three categories: terminological, structural and semantic. The interpretation of input data can also be used to categorize matching algorithms; [SE05] propose the following ones:

- element vs. structure level, describing the granularity of the match
- syntactic vs. external vs. semantic, describing the interpretation of the match

All categories (input and interpretation categories) use different kinds of techniques that will be explained in the following (see Figure 2.5 for an overview).

Mapping on the *element-level* implies that the elements (concepts, tables) are considered independently from their structure, i.e. the relations to other entities are not taken into account. In contrast, *structure-level* approaches analyze the structure of the entities when computing a mapping configuration. *Syntactic* techniques interpret the input data only considering the structure of the entities, while *external* methods use external knowledge like a thesaurus or human input. If methods exploit the meaning of the data, they are called *semantic* approaches. In the following, we want to give some examples for element- and structure level techniques illustrating the big variety of different possibilities to find a mapping configuration.

Element-level techniques may base on strings, languages, constraints, linguistic resources or previous alignments; additionally, upper level ontologies are taken into account (external techniques).
String-based techniques are used to compare and to map names on account of the string similarity between them. Several methods can be used to determine the string similarity; among them are prefix and suffix tests, distance functions like the edit distance and the n-gram measure.
In contrast, *language-based* techniques do not consider names as strings but as words of a natural language. Names gets tokenized by parsing them using blanks, points, digits etc. as separators. Additionally, tokens can be analyzed to find their basic forms, i.e. infinitives of verbs or the singular instead of the plural, what improves the comparability of the tokens. Some matching algorithms also include the elimination of stop words, such as articles, prepositions, conjunctions and so on.
Constraint-based techniques consider internal constraints, such as data types, value ranges and the number of attributes, subconcepts or keys. The closeness of data types or the cardinality of attributes may be determined to compare entities, for instance. The key characteristics of two entities can also be used to calculate a similarity value, i.e. if the (unique) keys of two entities match it is very likely that the entities describe the same concept.
Linguistic resources are common thesauri like WordNet [Wor10] that are used to detect synonyms or hyponyms.
The *reuse of alignments* that have been produced in previous matching processes is very useful, because a matching task might be similar to a previous one, especially if the ontologies describe the same domain. The saving of time can be very high.
Upper level formal ontologies are not used yet, but they could be used as additional external resources to capture the semantics of entities or to add structure to poorly structured ontologies.
Element-level techniques are also name schema-based matching methods.

Structure-level algorithms consider different kinds of information while searching for a mapping configuration. We can distinguish between graph-, model- or taxonomy-based techniques and the use of repositories of structures.
Graph-based techniques use graph algorithms to determine similarities between nodes of a labeled graph representing the database schema or the ontology. Graph matching algorithms determine a mapping, which minimizes a specific distance function, that considers children, leaves or relations. The similarity of inner nodes is based on the similarity of children/leave

nodes or already mapped concepts.
Taxonomy-based techniques again use graph structures but they examine only “is-a” relations. Concepts linked with such a relation are similar, hence neighbored concepts may be also similar. An exemplary rule considers sub/superconcepts: If sub/super-concepts are similar, it is very likely that the compared concepts also match.
Model-based techniques exploit the semantic interpretation of the input ontologies and apply methods like propositional satisfiability or description logics reasoning techniques.
The repository of structures stores all ontologies and pairwise similarities between them, but there are no alignments included. The goal is to find similar ontologies for each input ontology, such that an alignment can be reused (which is then extracted of the repository of alignments) or a new matching task between the input and the similar ontology can be initiated. To gain an effect of this repository, the similarity function that compares two ontologies has to have a lower cost than the matching process.

The classification of [SE05] only refers to schema- and structure-based matching approaches. Whenever schema information is not given satisfactorily or if the schemas to be mapped use different terms to describe the same real-world-concept, it is useful to consider instances. On the one hand, instance-based techniques can be used to improve the effectivity of schema-based approaches; on the other hand, instance-level techniques can be used on their own.

[RB01] propose a classification of instance-based methods by dividing them into linguistic- and constraint-based methods. *Linguistic-based* approaches use information retrieval techniques to gain information about word frequencies and combination of words. The use of value ranges, data types or the appearances of character patterns characterize the *constraint-based* approaches.

In [ES07] the classification of [SE05] is slightly extended by adding an extra kind of input category for ontology instances, which is named extensional. Additionally, *data analysis and statistics* is introduced as a new group of matching techniques which use (a sample of) the instances to derive patterns or to calculate frequency distributions.
Extensional matching techniques are also named instance-based matching methods.

An overview of the whole classification schema can be found in Figure 2.5.

2.7 Evaluation of Matching Systems

To determine their quality, matching systems need to be tested and evaluated. For this purpose it is helpful to use a well-defined test set. In the following, the different types of evaluation and the demands made on test sets are presented.

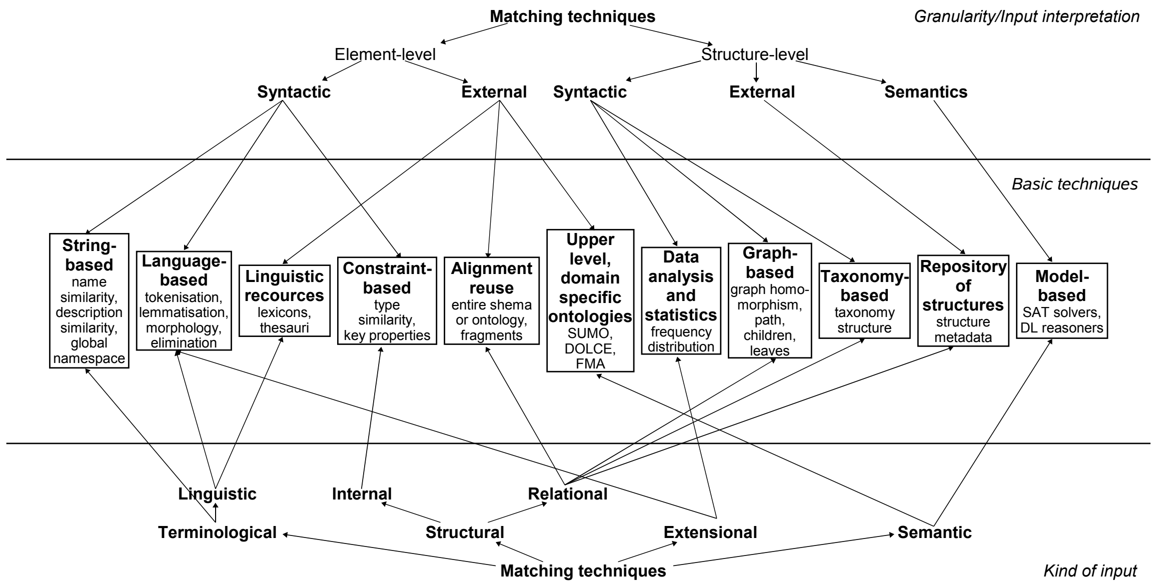


Figure 2.5: Classification of Matching Approaches, taken from [ES07]

2.7.1 Types of Evaluation

There are a lot of reasons for testing and evaluating matching systems. Independent of the particular evaluation design, there are several types of evaluation (see [ES07]): competence benchmarks, comparative evaluation and application-specific evaluation.

- *Competence benchmarks* are one possibility of evaluating systems. A benchmark contains a set of well-defined test cases, which provides the basis for determining the quality of a system. The single tasks are usually designed to test a particular aspect or method of the matching systems and the overall quality of the matching systems can be determined by observing the quality through all tests. The goal of this kind of evaluation is to find out the weak and strong points of a system and to improve the system.

- A *comparative evaluation* has the aim to compare systems by letting them execute the same tasks and determine the best system. For this purpose it is very important to define clear evaluation rules and the test data should be available shortly before the evaluation phase, such that systems can not easily adapt their systems parameter according to the tasks. A comparative evaluation can also be done by using a benchmark test series. The aim of this evaluation is to get an overview of the quality within the whole field.

- *Application-specific evaluation* is quite self-explanatory. Systems produce results for a specific application, e.g. proposed by a company that wants to find the best system for their specific problem. This kind of evaluation can also be used within a comparative evaluation.

2.7.2 Rules

Independent of the type of evaluation, the evaluation framework should follow a few principles to ensure a clear evaluation procedure. In [ES07] several general aspects are defined:

- systematic procedure: matchings tasks have to be reproducible and comprehensible, and the execution has to be comparable.
- continuity: the matching tasks should be executed continuously to observe developments and improvements.
- quality and equity: the evaluation rules have to be defined very exactly and the quality of the ontologies should be as high as possible. Additionally, no kind of matching systems may be privileged.
- dissemination: the benchmark and the results should be available publicly.
- intelligibility: the analysis of the results should be understandable, i.e. the reference alignments and the alignments produced by the systems should be available.

These rules should be generally taken into account when designing a particular evaluation framework.

2.7.3 The Data Set

The two most important things in any evaluation are the data sets and the used evaluation measures. The data set can be chosen or designed according to the different factors that influence the matching process (see Figure 2.4), i.e. input ontology, input alignment, parameters and resources, output alignment and the matching process itself. In the best case, the different evaluation tasks also differ in these dimensions to cover the "whole spectrum of matching" ([ES07], p. 198). In the following the different aspects will be examined according to their influence on the evaluation process.

- The *input ontologies* are the ontologies that need to be matched. These ontologies might differ in their language and the described knowledge (see also Section 2.4 for the different kinds of heterogeneity), and the number of ontologies might be two or more. Most systems focus on the matching of two ontologies, such that most evaluation frameworks focus on this problem, or the systems divide bigger matching tasks into smaller ones that only match two ontologies.

- A matching process can use an *input alignment* to enhance the matching quality. The input alignment might be produced by another matching process (e.g. using different methods) or could contain user-defined mapping candidates. There are two important characteristics: the alignment might be changed, updated or supplemented by the the matching process and the alignment may contain 1:1, 1:n or m:n mapping candidates. Most systems do not require or allow input alignments and for evaluation purposes it might be reasonable to perform the matching process without any input alignment to focus on the performance of the methods used within the system.

- *Parameters*, as well as *resources*, also influence the matching process. Resources might be oracles, thesauri like WordNet or catalogs like Google. User inputs also fall in the same category; users might verify matching proposals or support the system with their background knowledge. For evaluation purposes, one has to decide, if the use of external resources is allowed or not. The use can also be limited such that the resources can be incorporated but they must not be tuned according to the matching task.
  Most systems use a set of parameters to adapt the matching process according to the present matching task. Parameters might include weights for each single matching method or for a group of methods, thresholds and so on. In an evaluation process it is important to decide, if parameters can be changed for every matching tasks or if they should be equal for the whole set. For competitive benchmark tests it might be demanded to use a fixed parameter set whereas competence benchmarks aim to test the quality of a single system or a single matching method. For the latter case the parameters can be adapted such that the methods reach the maximum quality. It is also useful to vary the parameters to observe the stability of matching method according to different parameter sets.
  Some systems also provide the possibility to train themselves by using a sample data set, which influences the matching results and the quality as well. Providing a sample set for training might be very useful for systems that use machine learning techniques, because otherwise there might be a disadvantage for such systems.

- The *output alignment* can be characterized by many aspects: multiplicity, justification, relations and strictness. Multiplicity describes the cardinality of the mappings included in the alignment, 1:1, 1:n or n:m or also denoted as one-to-one, injective or total. A justification should explain the alignment in the case that a non-standardized alignment format is used. The relations between the entity pairs included in the alignment can be different; entities might be connected by an equivalence relation $=$, a subsumption relation $\leq$ or a incompatibility relation $\perp$. Additionally, a value or confidence measure illustrating the confidence of the mapping assignment can be included in the alignment (strictness). For competitive evaluations it is very useful to use a common alignment format (such as the one proposed in [Euz06]).

- For the *matching process* itself some aspects can be important. The space or time available for producing an alignment might be limited (resource constraints). As described before, entities may be concepts, attributes, relations or instances, and all these entities potentially might appear in an alignment. For evaluation purposes it is important to define which entity pairs can be included in the alignment, e.g. only concepts, everything besides instances, attribute-to-concept mappings are also possible etc.

2.7.4 Evaluation Measures

According to the rules and regarding the characteristics of the data set a matching system produces an alignment for the proposed matching tasks. The resulting alignment now has to be examined to determine its quality. For this purpose a reference alignment is needed, which can either be present in the used alignment format (for automatic evaluation) or the user has to create an alignment manually.
In the following some evaluation measures are presented.
Precision and *recall* are the most conventional evaluation measures. They have been developed for the field of information retrieval to express the quality of a search result. Precision expresses the accuracy and recall represents the completeness. Both measures have been adapted to ontology matching, the precise definitions are given below (adapted from [ES07], p. 206):
Given a reference alignment A^* and an alignment A, the precision $P_{A^*}(A)$ is defined as the number of correct found correspondences divided by the total number of found correspondences, more formally:

$$P_{A^*}(A) = \frac{|A^* \cap A|}{|A|} = \frac{\text{true positives}}{\text{number of correspondences}} \tag{2.14}$$

The recall value represents the number of correctly found correspondences divided by the number of existing correspondences, i.e.

$$R_{A^*}(A) = \frac{|A^* \cap A|}{|A^*|} = \frac{\text{true positives}}{\text{number of existing correspondences}} \tag{2.15}$$

When comparing systems among each other, a combined measures could be helpful. The *F-Measure* combines precision and recall by calculating their harmonic mean (see [ES07], p. 207):

$$F_{M_{A^*}}(A) = \frac{2 \times P_{A^*}(A) \times R_{A^*}(A)}{P_{A^*}(A) + R_{A^*}(A)} \tag{2.16}$$

Another measure combining precision and recall, which expresses the number of error corrections that are needed to transform A into A^*, is the *overall* measure. It can be calculated by

$$O_{A^*}(A) = R_{A^*}(A) \times \left(2 - \frac{1}{P_{A^*}(A)}\right) \tag{2.17}$$

The overall measure determines a value in $[-1, 1]$ and a negative value shows that the alignment does not have a high quality.
In literature, some more measures can be found, e.g. relaxed [EE05] or semantic [Euz07] precision and recall.

2.8 Summary

In this chapter we presented an overview of the basic principles in the field of ontology matching. The Semantic Web and ontologies have been introduced, including the detailed description of the RDF and OWL. Furthermore, the matching problem itself has been introduced and a classification of matching approaches has been presented. The variety of different matching methods is very huge, such that there are many possibilities to build a matching system. Some example matching systems and algorithms are presented in the next chapter. Finally, this chapter described the principles of matching system evaluation. There are already some benchmarks, which are also presented in the next chapter.

3 Related Work

The field of matching heterogeneous information sources has been widely addressed in the last decades, hence there is a lot of related work. In the Section 3.1 the most important approaches are presented; the focus lays on the description of the instance-based matchers and similar approaches are summarized. Afterwards in Section 3.2, the existing benchmarks for testing ontology matching systems are described. The chapter is concluded with a short summary.
An introducing article can be found in [Zai08b].

3.1 Matching Systems

In literature, there are many matching systems and single matching algorithms, which either use schema, structure- or instance information or a combination of them. In the scope of this thesis, the instance-based methods are most interesting such that we mainly focus on the description of such methods. First of all, a few schema- and instance-based matching systems are presented in Subsection 3.1.1, whereas we mainly focus on the presentation of COMA++, because, at present, it is the most complete matching tool. Subsection 3.1.2 gives an overview of instance-based matching methods that mainly make use of machine learning techniques or duplicates.

3.1.1 Schema- and Instance-Based Systems

COMA++

COMA++ is currently one of the most complete matching systems. The predecessor, COMA (COmbining MAtch algorithms), has been published in 2002 [DR02] and was developed for matching simple schemas. Coma++ extends COMA by adding the possibility to import complex XML schemas (e.g. ontologies), by inserting additional matchers and by offering a

fragment-based match approach to cope with large schemas.
Each input file is transferred into a specific internal graph representation, which provides the basis for the matching process. The matching library contains the following matching methods:

- simple matchers, mainly based on element names: Affix detection, Trigram, Soundex, Edit Distance, Synonyms, Type, Reuse and Statistics
- combined matchers: Name, NamePath, NameType, NameStat, Parents, Siblings, Children, Leaves

Some of the simple matchers are explained in detail in Section 2.5. *Synonym* uses a name synonym table (created with the help of users) to calculate a similarity between two elements (synonyms have a similarity of 1.0, hypernyms one of 0.8).
Equally, *Type* measures the similarity of data types. *Statistics* compares the structural statistics of an element (like number of children or parents) by applying the Euclidean distance function.
The combined matchers are composed by various simple matchers or additional structural matchers. *Name* combines the element-based simple matchers to calculate an aggregated similarity of the elements. For this purpose, the names are preprocessed by applying tokenization and acronym/abbreviation extension.
NamePath concatenates the element names of all elements that are contained in a common path (given by relations or hierarchical structures) to a long string and applies the Name matcher on it.
NameType combines the values of the Type and the Name matcher; equally *NameStat* combines Name and Statistics. *Children* is another structural matcher that propagates the similarity of leaf elements (obtained by applying another hybrid matcher like NameType) to upper elements. The similarity of an element is calculated by using the similarities of its children, regardless of their types (inner element or leaf). In contrast, *Leaves* only considers the similarities of the corresponding leaves.
Parents (as well as *Siblings*) calculate the similarity of two elements by measuring the similarity of parental elements or by considering the neighbors on the same level.

As presented in [EM07], the matching library has been extended by some instance-based matching methods, which can mainly be divided into constraint- and content-based methods. The constraint-based methods include the following constraints:

- general constraints: average length, used characters, differentiation between numbers, letters and special chars
- numerical constraints: determine whether a number is positive or negative and what type the number has (integer, float), average value, standard deviation
- pattern constraints: check, if all instances follow a given pattern of a predefined pattern library

The content-based matchers compares the instance sets of two concepts c_1 of ontology O_1 and c_2 of ontology O_2 by pairwise comparing the instances using *sim*, one of the string similarity function mentioned before, e.g. the edit distance or the trigram measure. The content-based similarity is then calculated by using the following formula:

$$\text{similarity } (c_1, c_2) = \frac{\sum_{k=1}^{n} max_{l=1\ldots m}(sim(i_{c_{1k}}, i_{c_{2l}})) + \sum_{l=1}^{m} max_{k=1\ldots n}(sim(i_{c_{1k}}, i_{c_{2l}}))}{n+m},$$

where n is the number of instances i assigned to c_1 and m the number of instances of c_2. Since not all concepts may contain instances, a similarity propagation algorithm is proposed to calculate the similarity of concepts that do not have instances by measuring the instance similarities of child concepts.

Regarding the classification of [SE05] given in Section 2.6, COMA++ uses syntactic and external techniques on the element level, as well as syntactic methods on the structure-level. Additionally, the use of instances is classified as extensional, and data analysis as well as language-based techniques are used.

The concept- and structure-based methods of COMA++ are well tested and perform quite good (see OAEI 2006 tests [MER06]). Additionally, the supported graphical user interface is very intuitive and provides a huge functionality.
The two instance-based matchers cannot be judged fully, because the introducing paper does not clarify one of the most important things for these algorithms, which is the comparison of the constraint values. It is not clear, how constraints like patterns are compared and how the different constraints are measured together. In contrast to the constraints-based comparison, the content-based similarity calculation needs a huge computing effort, which may be a problem for large numbers of instances.

Quick Ontology Mapping

Quick Ontology Mapping [ES04] (short: QOM) is another hybrid matching system, which uses instances, schema and structure information to produce a mapping. The algorithm used by QOM includes six steps:

1. Feature Engineering
2. Selection of Next Search Steps
3. Similarity Computing
4. Similarity Aggregation
5. Interpretation
6. Iteration

In the first step, the *Feature Engineering*, the ontologies are transformed into RDF triples. The *Selection of Next Search Steps* aims to reduce the number of entity pairs that have to be compared. For this purpose, QOM uses a strategy like Random (selection of a fixed number

of pairs) or Area (consider pairs that are close to already matched pairs).
In the *Similarity Computing* step, for each pair many different properties are compared, among them are: concept label, concept URI, direct property set, properties of subconcept, use of same-as relations, domain and range. The instances are also considered, but they are mainly tested on equality or only the URIs are regarded.
In the *Similarity Aggregation* step, the different similarities are normalized using a sigmoid function, weighted and finally summed up. The similarities are *interpreted*, i.e. according to several thresholds mapping candidates are determined. Finally, the whole process can be iterated for up to 10 times.
Referring to the classification in Section 2.6, we can state that QOM is a schema- and instance-based approach with two ontologies as input. It makes use of element-level (syntactic and external) techniques, but also of structure-level (syntactic) methods.
The biggest disadvantage of QOM is the selection of next search steps, because potential mapping candidates might not take part in the further matching process. Furthermore, the number of different similarity values that are calculated within the similarity computation step is very high, such that only entities, that are similar in many parts, have a high overall similarity value.

Other Systems

In the past many different ontology matching systems have been developed; an overview can be found in [ES07]. RiMom [Jie04], ASMOV [JMSK09], Anchor-Flood [SA09], Falcon-AO [JHCQ05], OLA [EV04], Sambo [LT06], AROMA [DGB07], GeRoMeSuite [KQL07] with an additional instance matcher described in [QGK09] and SemInt, described in [LC94] and [LC00], are further hybrid matchers.

Systems, that mostly rely on the schema information, are e.g. CtxMatch [BSZ03], H-Match [CFM06], DSSim [NVVM06], Lily [WX09], SimilarityFlooding [MGMR02], OntoMerge [DMQ03] and Cupid [MBR01].

3.1.2 Instance-Based Systems

Glue

Glue [DMDH02], [DMDH04], developed at the University of Washington, is another instance-based ontology matcher, which does not use additional schema-based methods. The mapping process consists of the three steps: Distribution Estimation, Similarity Estimation and Relaxation Labeling.

During the *Distribution Estimation* step the joint probabilities of each possible concept pair out of the two input ontologies are computed using a set of learners. In detail, the process of distribution estimation for a pair $(c_1, c_2) \in O_1 \times O_2$ works as follows:

- Divide the set of instances of O_1 into two sets: one set $U_{O_1}^{c_1}$ contains all instances belonging to c_1, the remaining instances are included in the second set $U_{O_1}^{\overline{c_1}}$.

- Train a learner with these two sets, which decides whether an instance belongs to c_1 or not.
- Repeat the first two steps for c_2 and ontology O_2.
- For each instance of c_1 apply the learner trained with instances of c_2 and vice-versa, i.e. calculate the sets $U_{O_1}^{(c_1,c_2)}$ and $U_{O_1}^{(\overline{c_1},c_2)}$ and $U_{O_2}^{(c_1,c_2)}$ and $U_{O_2}^{(\overline{e_1},e_2)}$ respectively.
- Calculate the probabilities $P(c_1,c_2), P(c_1,\overline{c_2}), P(\overline{c_1},c_2), P(\overline{c_1},\overline{c_2})$, e.g. by

$$P(c_1,c_2) = \frac{|U_{O_1}^{(c_1,c_2)}| + |U_{O_2}^{(c_1,c_2)}|}{|U_{O_1}| + |U_{O_2}|}.$$

- The remaining three joint probabilities can be computed in the same way.

There are three types of learners: Content Learner, Name Learner and Meta Learner. The *Content Learner* uses the instance values to train a Bayes classificator. The *Name Learner* works similar to the Content Learner, but the name of the respective instance is used. The name of an instance is the concatenation of the concept names from the root to the corresponding concept. *The Meta Learner* combines both learners and calculates a weighted average of the two probabilities.

After calculating the probability distributions, the calculated values are combined with a similarity function, e.g. the Jaccard-coefficient or the MSP measure, in the *Similarity Estimation* process. The result of this process is a similarity matrix containing the similarity values of all possible concept pairs. In the last step, this matrix, domain-specific constraints, heuristic knowledge and relaxation labeling (a graph algorithm, [Hum83]) are applied to find the best mapping configuration, i.e. to find pairs of concepts that match with a high probability. Constraints concern e.g. the neighborhood of a concept ("Two nodes match if their children match") or the frequency ("There can be at most one node that matches concept c_1").

So, Glue makes use of extensional matching techniques, and uses data analysis and statistic methods. Additionally, Glue uses graph-based matching methods to exploit the structure of the ontology.

The process executed of Glue is very time-consuming, because for the training of the learners all instances are used. Furthermore, this may lead to an overfitting.

Automatch

Automatch [BM02] is another instance-based matching system and has been developed at the University of Fairfax. Originally, it is designed for matching databases schemas, but the used methods can be adapted to ontology matching tasks. Automatch uses knowledge bases which include example values to describe the attributes. For each attribute of an schema there is an attribute dictionary, which consists of possible instance values and probability estimates (scores). The probability estimates are calculated by applying a Bayesian learning algorithm.

The values of the dictionary are chosen by following one of three statistical feature selection strategies: mutual information, information gain and likelihood ratio. The attributes directories are used to match the two input ontologies using the following algorithm: first of all, all attributes of the schemas are matched against the attribute dictionary and an individual match score is calculated. The individual scores are summed up to obtain attribute pair scores, which provide a basis for the application of a minimum cost maximum flow algorithm, that finally determines an optimal mapping.
Summarized, Automatch makes use of extensional matching techniques, based on the language, and data statistics.
The disadvantage of Automatch is, that it is time-consuming and it does not exploit schema information at all. Hence, entity pairs having the same name are not recognized without performing the complex instance-matching process. Additionally, Automatch cannot work with entities or ontologies, that do not have instances.

Dumas

DUMAS (Duplicate-based Matching of Schemas) [BN05] is an instance-based matching approach developed at the Technical University of Berlin. It uses the existence of fuzzy/approximate duplicates to find mapping correspondences and does not use any meta information like attribute names. To use this algorithm, one has to make the assumption that at least a few duplicates exists; but DUMAS does not search for all duplicates, only the K most similar instances are considered. The algorithm works as follows: each instance is represented as a single string. To enhance the quality of the comparison, stop words are removed and the string gets stemmed. The string represents a bag of words, which is translated into the vector space model, i.e. each dimension contains a weight for a term of the global vocabulary (i.e. the set of all used words). The weights are calculated by computing a SoftTFIDF measure [MYC08], which modifies the common TDIDF [SFW83] measure by taking the natural logarithm of the frequencies. The vector representations are then compared by applying the cosine measure. To reduce time and space complexity, not all instance pairs are compared, but the Whirl [CH98] algorithm or another sampling algorithm is used. The instance pairs are ranked by similarity and the K most similar pairs are selected. The attribute correspondences are deduced from the K high confidence duplicates by following this algorithm: For each duplicate compare the different attribute values among each other using a normalized edit distance. The result is a similarity matrix, in which all values beneath a user defined threshold are set to 0. This matrix is then used as input for an algorithm solving a bipartite graph matching algorithm which outputs corresponding attributes.
According to the classification schema, DUMAS is an instance-based matching system using language-based techniques and data statistics.
The drawback of DUMAS is, that it depends on the existence of duplicates. The system cannot work without duplicates and if the number of duplicates is too low, the matching quality is rather bad. DUMAS does no use schema information at all, which could reduce the time-complexity, and enable the matching of entities that do not provide instances.

Others

There are several other approaches, that also use instance information as input of machine-learning techniques. [WES08] make use of a Markov Random Fields [Kin80] to classify concept pairs as matched or unmatched. The basis is provided by the instance set which is used to create term frequency vectors for each attribute of each concept. These attribute vectors are then compared using the cosine similarity and a vector containing the calculated cosine similarities represents the concept pair. Finally, the concept feature vector gets classified by using the Markov Random Field.
[ITH03] use k-statistics (as presented in [Fle73]) on sets of documents to determine the similarity of categories in multiple internet directories. FCA-Merge [SM01] maps ontologies which share the same instance set. Further instance-based matching systems can be found in [ES07]; among them are LSD [DDL00] (a predecessor of GLUE), T-tree [Euz94], iMap [DLD$^+$04] and Caiman [LG01].

3.2 Benchmarks

Currently, there are some frameworks for evaluating ontology matching systems which are introduced in the following. There are also a few benchmarks for instance matching purposes. Instance matching aims at finding similar instances between heterogeneous information sources, e.g. ontologies, and is not equivalent to the instance-based matching of ontologies. Nevertheless, we examine some instance matching benchmarks according to their adaption for ontology matching tasks.

3.2.1 OAEI

The largest and most popular framework for testing ontology matching systems is the one published by the OAEI (see [OAE09] for the OAEI 2009). Since 2005, this initiative has the aim to provide a basis for system analysis. The OAEI organizes an annual workshop (Ontology Matching - OM) which is incorporated with the International Semantic Web Conference. Different test scenarios including different real-world matching tasks, an instance matching track and a benchmark are provided on the organizations website around 5 months before the OM workshop. Everyone can execute all or a subset of the matching tasks and send the results to the organizers, who evaluate them. The results are presented and discussed in the OM workshop.
The most interesting part of this framework is the *benchmark* test series. The benchmark includes ontologies describing the domain of bibliography. The starting point of every single matching task within this benchmark is a reference ontology. In 2009 this ontology consists of 33 concepts, 64 properties (40 object properties and 24 data type properties), 56 individuals and 24 anonymous individuals. The reference ontology is modified using different transformations to create 50 other ontologies.
The transformations can be divided into six categories:

- name: spelling mistakes, replacement by random names, different naming conventions and the translation into another language
- comments: suppression, translation
- specialization hierarchy: expansion or flattening
- instances: suppression
- properties: suppression or restriction
- classes: expansion or flattening

Each of the modified ontologies has to be matched against the reference ontology and the reference alignment is given. The systems participating in the contest execute all tasks of the benchmark, compute precision and recall and present their results at the OM workshop at the International Semantic Web Conference.
The OAEI benchmark ontologies only contain a very limited number of instances and no transformations are executed on the instance level (besides suppression). Hence, this benchmark is not sufficient for testing instance-based matchers or systems using a lot of instance-based matching methods.

There is also an *instance matching* track at the OAEI 2009 including three different benchmarks, the IIMB benchmark, the A-R-S benchmark and the T-S-D benchmark.

- The IIMB benchmark is created using the ISLab benchmark described in Subsection 3.2.3. In this benchmark only the instance set is modified, schema and structure of the ontology remain the same. The number of instances is around 300, i.e. it is slightly higher than in the OAEI benchmark ontologies.
- The A-R-S benchmark consists of three ontologies which all describe the domain of scientific publications: the Rexa ontology, which has 4 classes and 18492 instances, the eprints ontology, which contains the same 4 classes as Rexa but less properties and 1130 instances, and SWETO-DBLP.
- The T-S-D benchmark also includes three datasets: TAP, SWETO-testbed and DBpedia 3.2, which contain a bigger number of classes (around 120-200) and attributes and, additionally, a huge amount of instances.

For each benchmark test, a reference alignment is given as well, which includes similar instances. To adapt these benchmarks for ontology matching tasks, a reference alignment containing entity correspondences has to be created. Additionally, the number of variances in the schemas of the ontologies in the A-R-S and T-S-D benchmark is not sufficient for performing extensive studies with ontology matching systems.

3.2.2 STBenchmark

STBenchmark [ATV08] dynamically creates test (named target) ontologies by getting a reference (or source) ontology as input and applying several transformations on it. Basically, it consists of two components: a basic set of mapping scenarios and a generator for mapping scenarios and source instances.

- The *basic set of mapping scenarios* includes transformations like simply copying the instances, generation of constant instance values for some attributes, flattening or expanding the hierarchy and so on, which are applied on the source ontology to produce a target ontology.
- The *mapping scenario generator* `SGen` gets, as input, a set of parameters like number of subelements or depth of nesting and creates a mapping scenario as a result. More complex scenarios are obtained by concatenating multiple mapping scenarios. The *source instance generator* `IGen` uses ToxGene ([BMKL02]), a template-based XML data generator. First, `IGen` creates a ToxGene template according to several parameters (like maximum length of string values) and the source schema. The input template contains information about the data ranges and the desired vocabulary and is used for randomly (according to a Gaussian distribution) generating data values.

Equally to other related work, no transformations are applied on the instance level. Furthermore, the use of artificial data is not that useful for testing all aspects of a matching systems, because they follow a more or less strict pattern and it is difficult to completely simulate a human creator. Additionally, no reference alignment is given which complicates the evaluation; thus, the benchmark is not appropriate for comparative system evaluation/-analysis.

3.2.3 IIMB

The Islab Instance Matching benchmark (IIMB) [FLMV08] is a benchmark created for performing instance matching tasks. The reference ontology contains 5 concepts, 17 properties and 302 instances. Equally to the OAEI benchmark, the reference ontology is modified several times to create test scenarios but the transformations only happen on the instance level. The instances are modified e.g. by inserting typographical errors, by transforming the structure or by doing logical transformation.

The reference ontology is quite small and the number of available instances is slightly higher than the one of the OAEI benchmark but not high enough for extensive evaluation of instance-based matchers. The transformations done on the instance level are very useful, but a combination with modified meta information would be necessary for obtaining an ontology matching benchmark.

3.3 Summary

This chapter illustrated some of the existing matching systems. The whole field is very huge, such that many different approaches exists. Most systems focus on the use of schema and structure information to find entity correspondences, although instances can help to capture the semantics of attributes and concepts. The presented instance-based matching systems have some drawbacks and can be improved. In the next chapter, we present two novel instance-based matchers, which exploit the instance information to find entity correspondences in different ways.

We also presented some evaluation benchmarks in this chapter. The number of existing ones is not very high and the quality can be enhanced as well, especially in respect to instance-based matching methods and systems. Hence, we developed an additional benchmark, which is presented in Chapter 6.

4 Instance-based Matching

Matching of heterogeneous information sources is a problem that appears in many different applications, in artificial but also in real scenarios. Especially in the scope of the Semantic Web, ontologies become more and more important and the problem of matching or integrating them is existent, too. The data in the Semantic Web is structured by the concepts and relations of the underlying ontologies, and the displayed data is stored as instances. It is obvious, that the amount of instance data provides a high information content. Most existing matching systems do not make (sufficient) use of the information given by the instance set, but solely rely on class and structure information. Some kinds of heterogeneity can not be solved by only considering the schema information, e.g. if labels are chosen very subjectively or if they are meaningless. This lack motivates the development of instance-based matchers. In the scope of this thesis, two novel instance-based matchers are presented.
At first, we introduce a matcher that makes use of regular expressions or a catchword list to describe the instances of a concept. This provides a basis for creating a vectorial representation for each concept, the RECW vector. These vectors are compared by applying a similarity measure and an alignment is produced. The second approach also tries to characterize the concepts by regarding their instance sets, but in this case a set of predefined features is determined. Depending on the data types of the attributes, date, numerical or string features are calculated. The features provide the basis for a pairwise comparison between attributes of different ontologies. The attribute similarities are then used to compute a concept similarity.
This chapter is organized as follows:
In Section 4.1 a matcher using regular expressions or catchwords is presented. A second matching approach, which is based on the calculation of a feature set, is presented in Section 4.2. A short summary concludes this chapter.

4.1 Using Regular Expressions to Describe Instances

Instances provide a huge amount of data which should be used for matching ontologies. The difficulty is to process the information such that it characterizes the associated concepts as well as possible. Additionally, the comparability of different instance sets should be facilitated. The most obvious approach is to compare the instances directly, but this uses a lot of time, space and computing power. The approach presented in this section tries to represent a set of instances by one single value. Preliminary thoughts have been published in [ZSC08], the whole work can be found in [ZC09a].

Regular expressions seem to be suitable for this purpose. The best way to find one regular expression characterizing a set of data values would be to dynamically infer a regular expression that fits to all the instances. Unfortunately, until now it is very difficult to infer regular expressions on the basis of positive example values for an alphabet of a regular size. In [BGNV08] an approach is presented, which is able to infer deterministic regular expressions from a set of sample values, but the process is still very time-consuming. Additionally, whenever a sample is used, there is the danger of overfitting or the sample set might not be representative, such that the regular expression is too general. Hence, we decided to make use of predefined regular expressions, i.e. for each domain a set of regular expressions describing typical data values is created. This implicates an expenditure of time before performing a matching task, but the regular expressions can easily be reused in following matching processes or in other matching systems.

Nevertheless, it might be difficult to create appropriate regular expressions; either a domain expert might not be an expert for regular expressions, or the expert for regular expressions might not be familiar with the domain. To facilitate the process, we provide a second approach which makes use of catchwords. Catchwords represent data values, that are often assigned to concepts or attributes of the specific domain, i.e. they ideally represent the set of the most frequent values.

Independent of the used approach (regular expression or catchword), a fixed, ordered list including the expressions is created. For each attribute of a concept, the instance set is extracted and the best fitting regular expression/catchword is assigned to this attribute. The assigned expressions are collected for all attributes of a concept and are represented together in a single vector, the *RECW* (Regular Expression or CatchWord) vector, i.e. for each concept there is one RECW vector representing its underlying instances. The RECW vectors of the involved ontologies are compared pairwise by applying the cosine measure; the resulting similarity matrix provides the basis for determining a mapping.
According to the classification schema presented in 2.6, this matcher is an extensional matching approach, that uses data analysis techniques.
This section is organized as follows:
First of all, regular expressions are introduced in Subsection 4.1.1. In Subsection 4.1.2 the first approach is explained and the creation of the RECW vectors is described in Subsection

4.1.3. Afterwards, the second approach is presented in Subsection 4.1.4. In Subsection 4.1.5 the general matching process is clarified and the creation of candidate mappings is explained in Subsection 4.1.6. Evaluation and discussion of this approach can be found in the separate evaluation Section 7.1.2.

4.1.1 Regular Expressions

For our approach we need regular expressions which represent a kind of common denominator of a set of strings, i.e. instances in our case. The goal is to find a regular expression (RegEx for shorthand) that fits to the majority of instances included in a set. In general, there are different ways to compose regular expressions. We use the syntax described in [Hab04] to represent the regular expressions and for a better understanding of the example in Figure 4.1 we give some examples in the following (also see [ZC09a]):

- The regular expression

  ```
  R.*
  ```

  matches any string that starts with "R", followed by an arbitrary number of any character or digit (the dot denotes any character and the asterisk denotes that this symbol can appear arbitrarily often), it would match e.g. "RegEx".

- Regular expressions can be composed in many different variations. The expression

  ```
  .*(((c|C)onference)|((w|W)orkshop)).*
  ```

  would match any string in which "Conference", "conference", "Workshop" or "workshop" appears.

- For convenience, there are many abbreviations, e.g. "\d" for any digit and "\w" for any character (small or capital) or digit, and special symbols like "\s" for a whitespace. An expression, that makes use of these abbreviations, is

  ```
  [\w-\.]+@([\w-]+\.)+[\w-]{2,4}
  ```

  which describes email addresses. "[...]$\{2, 4\}$" denotes that the expression in the squared brackets has to occur twice at minimum and 4 times at maximum.

4.1.2 First Approach - Predefined Regular Expressions

As explained before, regular expressions are a good formalism to characterize a set of values, but the automatic inference is difficult. Hence, this matching approach uses a list of predefined regular expressions. The list is created manually by a domain expert, which has ideally experience in the domain the ontologies describe and in the construction of regular

expressions; otherwise two or more experts have to work together. It is also thinkable to search for predefined regular expressions in the web or - which is the long-term goal - to reuse lists of former matching processes. There might be values that are very unspecific, such that a list of regular expressions should always contain very general regular expressions that fit to most of the instances. But this poses a problem since an instance might fit to several regular expressions. For this purpose, the lists get ordered, from very specialized at the beginning to very general at the end. For each attribute, we extract k instances, and for each instance we iterate over the regular expression list. The first regular expression to which the instance fits, is assigned to the instance. The regular expression that fits to the majority of the instances of an attribute is assigned to the attribute.
In ontologies, instances assigned to a concept are resources that have a unique URI or identifier, which in most cases also includes information on the semantics of the instances. In some cases, the name of the instance is (also) represented as a `rdfs:label` attribute, but in other cases the label is only encoded in the instance name. The following two ontology snippets should clarify this observation.
In this snippet the instance name `a43836633` is senseless and the actual name of the instance is included in the `rdfs:label` tag.

```
<Proceedings rdf:about="#a32071928">
<rdfs:label>Proceedings of the First European Semantic Web Symposium</rdfs:label>
<rdfs:year>1998<rdfs:year>
</Proceedings>
```

Another possibility to model the same fact would be:

```
<Proceedings rdf:about="#ProceedingsOfTheFirstEuropeanSemanticWebSymposium">
<rdfs:year>1998<rdfs:year>
</Proceedings>
```

In contrast to the first possibility, in the latter case the name of the instance would not be considered as an attribute instance value and consequently would not take part in the similarity calculation process. Thus, we decided to create an additional attribute named `label` for each concept, that does not have such an attribute by definition. The instance names are then assigned to this new attribute and join in the feature calculation process as every other instance.

4.1.3 Arranging Lists and Creating Vectors

As explained before, the regular expressions describing the instances of a domain are created manually by experts and collected in a list. Since all instances should only get assigned to one regular expression, i.e. the first fitting one, the list needs to be ordered such that very special regular expressions are at the top and very general expressions are on the bottom of the list. In our case “very special” means that the fewest number of instances matches this expression. For this purposes we extract a sample set of instances from one ontology and “train” the list using them. The used procedure is described in Algorithm 1.

```
input  : unordered list $u$ containing $d$ RegEx
output: ordered RegEx list $l$
*init. d-dim array with zeros*
int[] counter=zeros(1,d);
*count RegEx matches*
forall ontology $o$, concept $c \in o$, attribute $a \in c$ do
    insert $k$ attribute instances $i \in a$ in set $I_a$;
    forall $i \in I_a$ do
        for $r = 1, ..., d$ do
            if $r$-th RegEx of list $u$ matches instance $i$ then
                counter[r]++;
            end
        end
    end
end
list $l$ =emptyList;
/*sort list $l$ according to counter[]*/
for $r = 1, ..., d$ do
    find $m$ with counter[$m$] = max! and counter[m]$\geq$ 0;
    $l$.appendAtBeginning($m$-th RegEx of list $u$);
    counter[m]=-1;
end
return $l$
```

Algorithm 1: Arranging the regular expression list

First of all, for each regular expression a counter is initialized. For each attribute a a set I_a containing k instances is extracted. Then we iterate over the regular expression list u and compare the instances to the regular expressions. Whenever an instance matches a regular expression, the counter of the corresponding regular expression is incremented. After repeating this process for all attributes of the ontology the list is ordered according to the counter values. The regular expression which describes the least instances, i.e. whose counter has the lowest value, is set to the top of the list. The following expression are ordered ascending concerning their counter values and as a result we obtain the ordered list l. Now, we can define a bijective function $\delta : \{1, ..., d\} \rightarrow \Theta$, where Θ is the subspace of regular expressions that are contained in l (and u), with

$$\delta(i) = r_i,$$

where r is the i-th regular expression of the ordered regular expression list l.
As explained before in Subsection 4.1.2, the ordered regular expression list is used to assign regular expressions to attributes. According to this process, we can now define a function $reg : A \times \Theta$ from the attribute space A to the space of regular expression with

$$reg(a) = argmax_{r_k}(counter_a[r_k]), 1 \leq k \leq d,$$

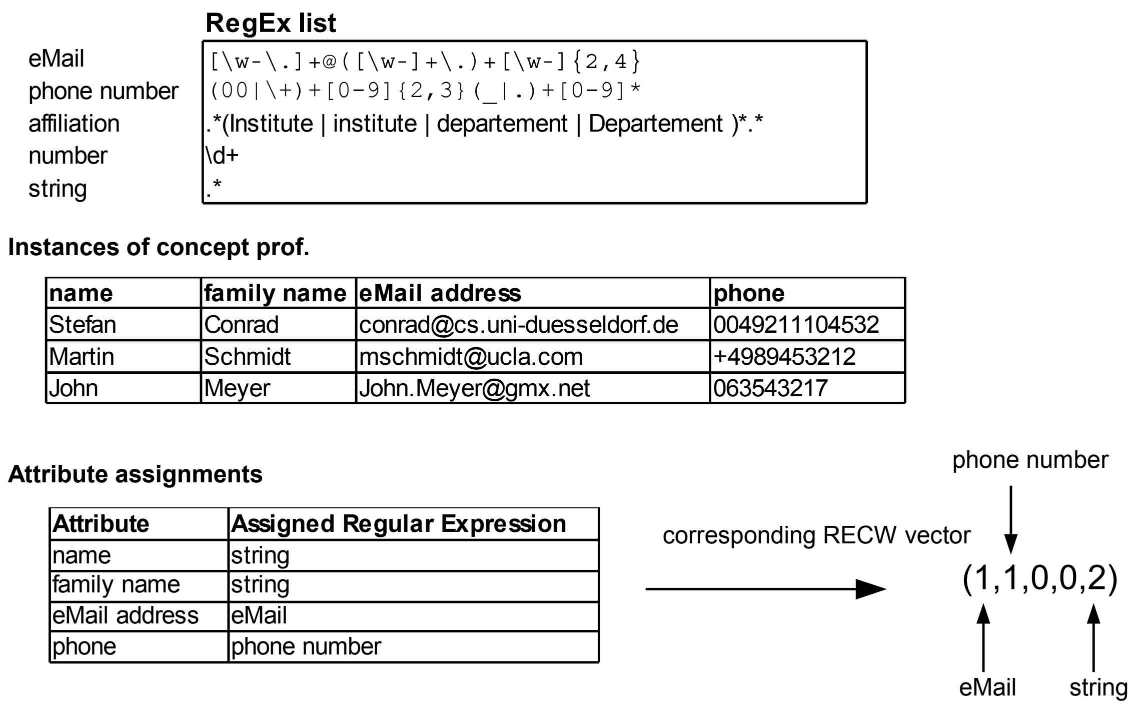
where a is an attribute of a certain concept and r_k the regular expression that is assigned to a (i.e. to the majority of examined instances belonging to this attribute). After determining the regular expression assignments, they are summarized for each concept and represented in a vector representation, the *RECW* vector. The RECW vector $v \in \mathbb{N}^d$, representing a concept c, is the vector whose components v_i $(i = 1, ..., d)$ contain the number of assignments from $\delta(i)$ (i.e. the i-th regular expression in the RegEx list l) to an attribute in c. Hence, the sum of all components of v equals the number of attributes in c. The algorithm of building the vector is clarified in Algorithm 2.

```
input  : concept $c$, ordered RegEx list $l$
output: RECW vector $v$
calculate $\delta$ and $\delta^{-1}$;
*calculate function $reg$:*
forall attribute $a \in$ concept $c$ do
    compare certain amount of instances with RegEx list $l$ to determine $reg(a)$;
end
*initialize RECW vector $v$:*
forall component $v_i$, $1 \leq i \leq d$ do
    $v_i = 0$;
end
*count allocated RegEx in accordant component:*
forall attribute $a \in c$ do
    $inc(v_{\delta^{-1}(reg(a))}, 1)$;
end
return $v$
```

Algorithm 2: Building the RECW vector

The process gets clarified by the example that is displayed in Figure 4.1.

In general, regular expressions that fit to most of the instances of an attribute are assigned to the attribute. In the case, that there is a very special regular expression (i.e. an expression at the top of the list), that describes many of the instances of an attribute, but less that $k/2$ (due to the randomly selection of the k instances, or due to spelling mistakes), the attribute would be assigned to the less specific regular expression. However, the major goal of our approach is to find the most specific regular expression for each attribute. For this purpose, we can use a weighted regular assignment approach. As before, the instances are matched to the regular expression list and the first fitting regular expression is assigned to



RegEx list

```
eMail          [\w-\.]+@([\w-]+\.)+[\w-]{2,4}
phone number   (00|\+)+[0-9]{2,3}(_|.)+[0-9]*
affiliation    .*(Institute | institute | departement | Departement )*.*
number         \d+
string         .*
```

Instances of concept prof.

name	family name	eMail address	phone
Stefan	Conrad	conrad@cs.uni-duesseldorf.de	0049211104532
Martin	Schmidt	mschmidt@ucla.com	+4989453212
John	Meyer	John.Meyer@gmx.net	063543217

Attribute assignments

Attribute	Assigned Regular Expression
name	string
family name	string
eMail address	eMail
phone	phone number

Figure 4.1: Creation of RECW vectors, an example

the instance, i.e. the counter of the corresponding regular expression is incremented. This process is repeated for all k instances of an attribute. Normally, the regular expression with the highest counter would be assigned to the attribute. In the weighted approach the counters are multiplied by a weight w_{r_i} which depends on the position of the regular expression in the ordered list l:

$$w_{r_i} = \frac{1}{i} \cdot d$$

with i being the position of r_i in the ordered list l and d being the total number of regular expressions. Using this weighting formula, the first, i.e the most specific regular expression, gets the highest weight and the last the lowest weight. The assignment of a regular expression to an attribute is then determined using:

$$reg(a) = argmax_{r_k}(w_{r_k} * counter_a[r_k]), 1 \leq k \leq d, \tag{4.1}$$

4.1.4 Second Approach - Transforming Catchwords

The building of regular expressions is the most precise way to characterize instance sets, but in some cases it might be difficult to determine them. On the one hand, there might not be an expert which has experience in both, the domain and in formulating regular expressions. On the other hand, it is very time-consuming to build complex regular expressions. To facilitate the matching process, we decided to expand the approach described before by offering the possibility to specify so-called "catchwords" instead of regular expressions.

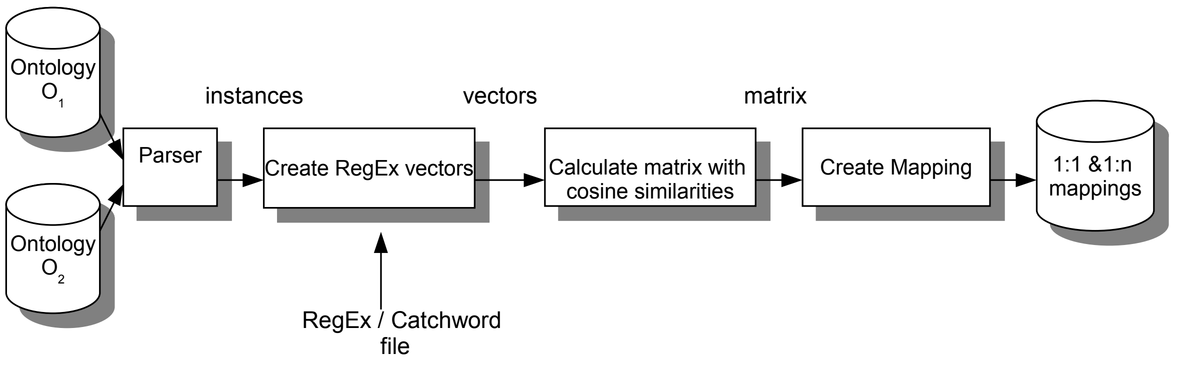


Figure 4.2: Instance-Based Matching Process

In our case, catchwords denote groups of strings $(s_1, s_2, ..., s_n)$, where the values s_i represent frequent values (strings or substrings) related to an attribute of the ontology. When creating the catchword list, the domain expert should think of values that are typical for the domain the ontology describes. In our example frequent substrings could be "@" or ".com" for an attribute like email, or the word "Institute" might appear in many instances describing the affiliation of a person. Values, that are assigned to the same attribute, form one string set (or catchword group) which is treated equally to a singular expression. Obviously, it is not always easy to specify frequent substrings for all attributes. The instances of an attribute like "name" generally do not contain common substrings, but there might be values, that appear more often than others, e.g. "Smith". The expert manually determines such frequent values. Alternatively, one can extract sample values for each attribute out of the instance sets, where the most frequent values should be selected. This process is also required for numerical attributes, because it is difficult to specify frequent values manually. Furthermore, each catchword list is automatically extended by values of standard attributes (with limited space of possible instances) like month or day.
The process of obtaining the RECW vectors remains nearly the same. The instances set of each attribute is matched against the catchword list. An instances matches a catchword group $(s_1, s_2, ..., s_n)$ if the instance equals one of the values s_i or if the instance includes one of the s_i as substring. Due to the lack of order in the catchword list, an instance might map to more than one catchword group, but this does not influence the process; it only means that the counters of more than one catchword group might be incremented. Apart from that, the process of building the RECW vector is equal to the one described in Algorithm 2 (just replace RegEx by catchword group).

4.1.5 General Process

The two approaches described before calculate the RECW vectors for each concept of the input ontologies. To determine an alignment between the ontologies, we have to compare the vectors. The general flow of the matching method is shown in Figure 4.2.

As input we need a set of k instances for each concept of the two ontologies O_1, O_2 that should be matched. Using these instance sets we compute the RECW vector sets $V_{O_1} = \{v_1, ..., v_n\}$ and $W_{O_2} = \{w_1, ..., w_m\}$ with the help of (ordered) regular expressions/catchword lists as described before. Each vector contains digits representing the number of attributes of a specific concept that are assigned to the regular expression/catchword group. Thereafter, the vectors are compared by calculating a similarity with the cosine measure:

$$sim_{cos}(v, w) = \frac{v * w}{|v| \cdot |w|}, \tag{4.2}$$

with v and w being two RECW vectors. The values contained in the vector do not differ very much (because the sum of them always equals the number of attributes which is not that varying) and in many dimensions the value is 0. Due to this, the Manhattan or Euclidean distance of the vectors would be rather small. The cosine measure better considers the different dimensions, i.e. the distance is smaller if the vectors share values > 0 in the same dimension and dimensions, in which only one vector has a value, are neglected completely; hence it produces better results for this application.
After calculating sim_{cos} for each $(v_i, w_j) \in V_{O_1} \times W_{O_2}$ we obtain a similarity matrix which provides the basis for the computation of an alignment.

4.1.6 Creation of Candidate Mapping

Using the similarity matrix with the cosine similarities, an alignment can be computed. The process is described in Algorithm 3, in which both 1:1 and 1:n mappings are considered. The process of finding 1:n mappings is explained in detail in Example 4.
The similarity matrix is taken as input and all concept pairs that have a similarity of t_{max} (a predefined threshold) or higher are directly inserted into the mapping set Map, these mappings are 1:1 mappings. The process of finding 1:n mappings is motivated by the following Example:

Example 4. *Considering our example ontologies in Figure 2.1 and 2.3, we see that a professor and his email address are modeled differently. The first ontology has two concepts, "professor" and "email", where professor contains three attributes "phone", "name" and "surname" and email is an object property with the attribute "address". If the same regular expression list is used as in Figure 4.1 the two concepts could be represented by the vectors* $v_{professor} = (0, 0, 0, 1, 2)$ *and* $v_{address} = (1, 1, 0, 0, 0)$*. The same fact is modeled in the second ontology within the concept "prof." which is represented by the vector* $w_{prof.} = (1, 1, 0, 0, 2)$*. The cosine similarity between these vectors is* $sim_{cos}(v_{professor}, w_{prof.}) = 0.73$ *and* $sim_{cos}(v_{address}, w_{prof.}) = 0.58$*. These values might be below a threshold and consequently, these concept pairs would not be considered in a mapping. For humans it is obvious that the two concepts "professor" and "address" map to the concept "prof.". Our algorithm is also able to detect such relations, because it also compares combinations of concepts by adding the corresponding RECW vectors. In this example "professor and email" have a common RECW vector which is* $v_{professor\&address} = (1, 1, 0, 1, 2)$ *and the cosine similarity to* $w_{prof.}$ *is* 0.93 *which indicates a mapping.*

The process of finding 1:n mappings works as follows. For each concept $c_i \in O_1$ a candidate mapping set CM_i, which contains all concepts $c_j \in O_2$ having a similarity between t_{min} and t_{max} to concept $c_i \in O_1$, is generated. Construct the power set P of all elements in CM_i to calculate all possible combinations of the concepts c_j of O_2. For each combination included in $P(CM_i)$ sum up the corresponding RECW vectors (the result is a vector w_{cm}) and calculate the cosine similarity between concept vectors v_i and w_cm. If the new similarity is above t_{max}, add the corresponding concepts to Map. The whole process has to be repeated for the second ontology. As a result, we obtain the final set of mapping correspondences.

```
input  : similarity matrix $M$
         with $i$ identifier for a concept vector of $O_1$ and $j$ of $O_2$
         $[i, j]$ cosine similarity of the vectors
         minimum threshold $t_{min}$
         maximum threshold $t_{max}$
output: set $Map$ of 1:n mappings
for $i = 0, ..., M.length() - 1$ do
    define set $CM_i$ containing mapping candidates
    for $j = 0, ..., M.length() - 1$ do
        if $[i, j] \geq t_{max}$ then
            $Map \leftarrow (i, j)$
        end
        else if $t_{min} < [i, j] < t_{max}$ then
            $CM_i \leftarrow j$
        end
    end
    construct power set $\mathcal{P}(CM_i)$
    forall $cm \in \mathcal{P}(CM_i)$ do
        sum up vectors for all concepts, whose identifier is in $cm$
        result: vector $w_{cm}$
        calculate sim $sc$ of concept vector $i$ and $w_{cm}$
        if $sc > t_{max}$ then
            $Map \leftarrow (i, cm)$
        end
    end
end
return $Map$
```

Algorithm 3: Finding 1:n Mappings

4.1.7 Finding Attribute Correspondences

The approach described so far only determines concept correspondences; attributes or relations are not considered. To find property correspondences, we use those concept correspondences that have a high confidence, i.e. a high similarity, and that are most probably a correct match. For each concept of such a high confidence pair, the properties are compared pairwise using the regular expression, they are assigned to, and a string-based similarity measure, e.g. based on the edit-distance, that compares the attribute names. One can limit the attribute comparison pairs to those, that have instances and hence are assigned to a regular expression, or extend the approach and compare all attributes of the concepts. For attributes, that do not contain instances, only the string similarity is calculated.

4.2 Combing Concept and Instance Features

The matching method described in the previous section aims at characterizing the whole instance set of an attribute by one single feature, i.e. the assignment to a regular expression or a catchword group. The instance-based matching method proposed in this section calculates a set of features, e.g. average length or frequent values, for all properties of each concept that provides instances. In this process we distinguish between dates, numerical and string-based attribute values and calculate specific features. Properties of different ontologies are then compared by calculating a similarity using the corresponding features. Finally, the property similarities are propagated to an overall concept similarity.
Preliminary work has been published in [ZC09b]. The work described in this paper has been extended by adding some extra features and by considering concept information, i.e. by calculating concept features as well to enhance the matching quality.
With regard to the classification of Section 2.6, the matcher presented in this section is mainly extensional and uses data analysis and statistic techniques as well as language-based methods. Furthermore, for the concept features linguistics resources are used and string-based matching methods are used.

The remainder of this section is organized as follows: First, the general process is described in Subsection 4.2.1; the process of extracting features is explained in Subsection 4.2.2. The different features belonging to the different data types are presented in the following subsections: Subsection 4.2.3 describes the concept features, Subsection 4.2.4 lists the numeric features, the string features are presented in Subsection 4.2.5 and the date features are described in Subsection 4.2.6. The process of comparing the features and determining an alignment is included in Subsection 4.2.7. An evaluation of this approach and thoughts on future work can be found in the additional evaluation Section 7.1.3.

4.2.1 General Process

Only concepts that contain instances are interesting for our approach. Each concept, that provides instances, is involved in the feature calculation process. Concept instances are split into attribute instances and for each attribute concept-based features and date, numeric or string-based features are calculated (depending on the type). Additionally, the resource URIs assigned to object properties are included in the process. The result is a feature cube which contains feature values for each property of each concept. If there are no instances for a single property, the values of the numeric, date or string features are set to "null" and only the concept-based features obtain values. This process is executed for both input ontologies; hence two feature cubes are created. To match the two ontologies, the feature cubes have to be compared according to the following procedure: As usual, the concepts of the two ontologies are compared in pairs, in which each concept of one ontology is compared to each concept of the other ontology. For each concept pair the corresponding slice of the feature cube is extracted. The slice is then divided into single feature vectors that belong to one property. The properties of one concept are compared with all properties of the other concept by applying the similarity function described in Section 4.2.8. It is important to state that only similar data types are compared, i.e. numeric features are compared with numeric features and the same holds for date and string features. The result of the comparison process is a similarity matrix which contains similarity values for each pair of properties of each concept (besides the properties of different types; in these cases the similarity value is set to "null"). Using this matrix, the similarities on the property level can be used to obtain property correspondences and finally, the similarities can be propagated to the concept level, such that concept mapping candidates can be determined.

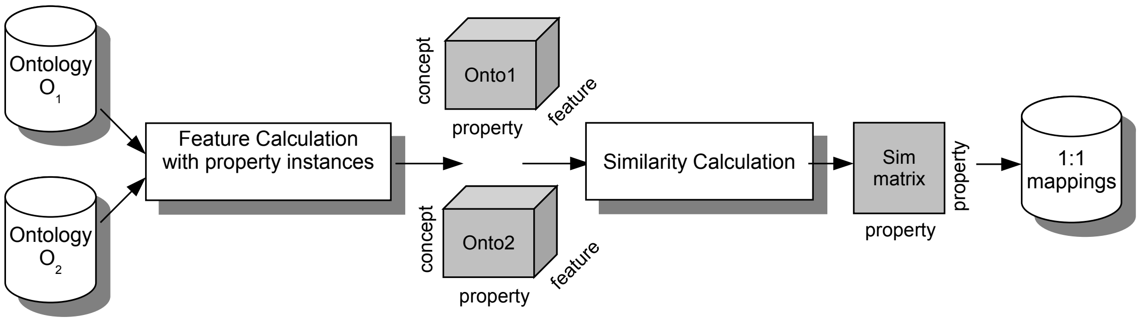


Figure 4.3: Matching Process

4.2.2 Extracting Features

Our matching approach focuses on the instances of an ontology, but adds several concept-based features to exploit the expressiveness of the meta information. Normally, instances belong to whole concepts; for our purpose the instances are separated according to the attributes they describe. It is easier to capture the semantics, i.e. to calculate features, for a single attribute, that has similar instance values, than for all instances of a concept at once. Hence, for each attribute of a concept we obtain an instance set, which provides a basis for the feature calculation process. Due to this procedure, it does not matter if a concept instance only contains attribute instances for some attributes, because they are treated independently anyway.

Our definition of attribute only includes `datatype properties`. Since concepts often contain many `object properties`, we decided to include them in the feature calculation process as well. Object properties denote the relation of two concepts and their instances respectively, i.e. the value of an object property is another resource. The resource URI is treated as the instance of the object property. In the following, datatype and object properties are subsumed as `properties`. The pseudo code Algorithm 4 clarifies the process of feature extraction.

For each concept of an ontology the corresponding property set P_c is extracted. For each $p \in P_c$ k instances are extracted, independently of the other properties of the concept. Thereupon, each instance set is used to calculate the different features. First of all, several

```
input  : concept $c \in C$, threshold $k$
output: feature vector set $F_c$
Extract all properties of $c \rightarrow P_c$
forall properties $p \in P_c$ do
    extract $k$ property instances $I_p$;
    FeatureCalculation($I_p$) $\leftarrow$ feature vector $f_i$
    $F_c \rightarrow f_i$
end
return $F_c$
```

Algorithm 4: Extracting Features

concept features (including property name etc.) are determined. Depending on the type of the instances, we calculate numerical, date or string features as well. If there are no instances for a property, only the concept features are determined. Finally, there is a feature set for each property which provides the basis for the propertywise comparison of two concepts.

4.2.3 Concept Features

Previous work showed, that meta information has to be considered if the matching quality should be high for different matching scenarios, because ontologies do not always contain instances for all concepts. In most of the cases the comparison of concept and property names or comments provides good results, if the ontologies are written in the same language. Naturally, the use of language is always subjective; people use a different vocabulary and there are a lot of synonyms and homonyms to be considered. However, the use of concept-based matching methods is essential to find mappings if no other information (like instances) is available.

The matching approach proposed in this section can be integrated in a more complex system (see Chapter 5) to increase the influence of meta information-based methods. By now, we focus on a few important features extracted out of the meta information to complete our feature set. Among them are:

- property name
- name of concept and name of father concept (if available)
- property comments; only key words; elimination of stop words, stemming; if no comment is available, extract definitions from thesauri
- synonyms of the property name

The *name* of the property is the most obvious meta information that can be used for comparison. If two properties have the same or a similar name, it is very likely that they describe the same real world fact (except in the case of homonyms).

The surrounding context of a concept might help to limit the semantics of the property (especially for synonyms). Since the features are calculated for each property of a concept, the

value of the *concept and father concept name* feature is the same for all properties of each concept.
Ontologies often include *comments* expressed in natural language to describe the meaning of an entity, e.g. of attributes. If property comments are available, keywords are extracted by eliminating stop words and using a stemming algorithm. Alternatively, a property description is extracted using a thesauri (e.g. WordNet).
As described in Subsection 2.4, *synonyms* are a well known problem in the matching process. For this purpose, the synonyms of the property are determined using WordNet [Wor10].

4.2.4 Numerical Features

Attribute values are often expressed in numerical values; object properties are always treated as string features. On the one hand, it is easier to calculate features out of a set of numerical values because there are a lot of different measures (from statistics) that can be applied. On the other hand, the used alphabet is limited to eleven elements (ten digits and a point), hence the range of different values is smaller in contrast to string attributes. Besides this, numbers do not contain obvious semantics like attributes expressed in natural language, i.e. the semantics of numeric attributes is more difficult to capture if no meta information is implied. However, we try to bridge this semantic gap by calculating various features that describe the numerical data set as detailed as possible. Additionally, some features using the meta information of the corresponding concept and its attributes are determined (see Subsection 4.2.3). In summarization, the following features are calculated for each attribute that includes numerical instances:

- average value
- range, i.e. minimum and maximum value
- variance
- coefficient of variance
- set of frequent values
- most frequent number of digits (in total and divided into places before and after the decimal point)

The *average value* is the most obvious feature that can be calculated out of a set of numeric values. In combination with the value *range*, it gives a detailed overview of the set. Both values can be a strong hint for similarity, e.g. the range of marks is quite similar in general. But minimum and maximum value could also be outliers occurred by typing mistakes or accidentally using different scales. In this case the *variance* could be a useful measure.
If two instance sets include the same data but it is expressed in different scales, average value and range might differ significantly. For this purpose, the *coefficient of variance* is implicated in the feature set. The coefficient of variance is defined as the standard deviation divided by

the average value and provides good results for data sets expressing similar data in different scales.
It is very likely that two instance sets describing the same real world fact share some values. Several matching systems search for duplicates and use this information for the matching process. Our approach extracts the most *frequent values*; values are “frequent” if x percent of the instances in this set take this value. The threshold x can be defined by the user or it can be set to a fixed number of frequent values that should be returned (if possible).
The last feature, the *most frequent number of digits*, is divided into two parts: first of all, all digits are counted independently of their position; additionally, the number of digits before and after the decimal point is determined. On the one hand, this feature expresses the accuracy of the values (the decimal places), and on the other hand, it says something about the average length of the values. As said before the range may be influenced by outliers, and the length of the average value may not be the most frequent length within the instance set.

4.2.5 String Features

In general, most of the instance values may be strings; especially the resource values of object properties are always treated as strings. In contrast to numerical values, the used alphabet is much bigger which results in a huge variety of instances. String values are normally expressed in terms of a natural language (excluding values like DNA sequences e.g.), which means that the values might be subjective. The same address can be expressed using a different format or using abbreviations; a bigger difference might occur if strings represent a comment or a rating a person has composed to describe a book or a film. As explained before, there is also the problem of using synonyms and homonyms and the used vocabulary might differ from person to person. Furthermore, spelling mistakes might occur. Due to all these facts it is quite difficult to compare string values in some cases. Our approach tries to compare sets of strings by reducing them to some features:

- average length and variance
- common substrings
- set of frequent values
- average number of words (or tokens) and variance
- set of used special chars
- ratio of chars and numbers
- regular expression

The length of a string might be characteristic for certain attributes. Descriptions are longer than names, but names are usually longer than abbreviated strings describing the sex oder another status for example. By calculating the *average length* and the *variance* we obtain two useful features.

Common substrings are very interesting; they define a kind of pattern, which most of the instances follow. For our purpose common substrings are strings with a minimum length of 3, that are included in x_c percent of the values. The limitation of the length on a value greater or equal than three has been done due to the fact, that the frequency of substrings of length two is too high and too expressionless. If an instance value consists of more than one word, common substrings are searched for each word of length ≥ 3.
The *frequent value* feature is similar to the search for duplicates that many matching systems use. The definition of frequent is equivalent to the one given for the calculation of frequent numerical values.
For comparison it is also important to determine the *average number of words*. Names might be described in two words, comments are much longer. It is also very important to get an overview of the length distribution, so the *variance* is calculated as well.
String values often contain more than just simple characters, namely digits and or special chars. Both give a good hint on the semantics of the property: *special chars* like "@" are included in email addresses, normal addresses include points and digits etc. In fact, special chars are frequent substrings of length 1 but they are not included in the common substring set.
To determine the average number of used digits, the *ratio* between chars and number is calculated.
The last feature tries to characterize the instance set by creating a *regular expression*, that fits to most of the instances. The regular expressions are very simple and do only include the following symbols: c for one character, s for string (more than one character, at most one word), d for digit and the included special chars. The regular expression fitting the value "Madison Ave 234" would be "ssd"; a value like "zaiss@cs.uni-duesseldorf.de" is described by the regular expression "s@s.s-s.s".

4.2.6 Date Features

In our previous approach (described in [ZC09b]) we focused on string and numeric features; dates have been treated as string, but the string features are not expressive enough to distinguish between different dates, because the average length and the used special chars are similar in most cases. Consequently, we decided to calculate special features for date values:

- day, month and year range
- frequent days, months and years

First of all, the date values are split into day, month and year. For each part of the date the range is calculated. Starting dates of new employees might always be on the beginning of a month and earnings payments either on the 15th or on the last day of the month. Using the range together with frequent values for day, month or year we can better distinguish between different attributes including dates.

4.2.7 Comparing Features

The previous subsections describe the different features that are calculated by using the instance sets of an properties. As a result we obtain two feature cubes (one for each ontology) which contain feature values for all attributes of all concepts. To match the two ontologies, the two cubes have to be compared in an appropriate way. For this purpose, the cubes are sliced according to the concepts and then divided into feature vectors describing single attributes. These feature vectors are compared using a similarity function. In general, there are a lot of different possibilities to calculate a similarity value between two vectors. In our case it is quite difficult to find an appropriate measure, because the values of the feature vectors are heterogeneous; they contain numeric and string values or sets of different values. Section 4.2.8 describes the similarity measures used for evaluation in this paper. The result of determining a similarity between all pairs of attributes is a similarity matrix, which is used to calculate property correspondences and subsequently, to propagate a concept similarity (see Section 4.2.9). Finally, the concept similarities provide a basis for computing a mapping (see Section 4.2.10).
Pseudo code Algorithm 5 gives an overview of the comparing process (O_1 and O_2 are the two ontologies to be matched).

```
input  : feature cubes $F_1$ of $O$, $F_2$ of $P$
output: Mapping $M$
Define set $F_{O_1}, F_{O_2}$ as the set of feature vectors of $O_1$ and $O_2$;
Define matrix $M_{sim}$;
forall concept $c_k \in O_1, c_m \in O_2$ do
    Slice the cube $\rightarrow slice_{c_l}$;
    forall attributes $a_{c_l} \in slice_{c_l}$ do
        Divide slice according to property $p_{c_l} \rightarrow$ feature vector $f_{p_{c_l}}$;
        $F_X \leftarrow f_{p_{c_l}}, X = O_1$ or $O_2$;
        forall $f_{p_{c_k}} \in F_{O_1}$ and $f_{p_{c_m}} \in F_{O_2}$ do
            $M_{sim} \leftarrow$ SimilarityCalculation($f_{p_{c_k}}, f_{p_{c_m}}$);
        end
    end
end
$M_{conceptSim}$ = SimilarityPropagation($M_{sim}$);
DeterminationOfMapping ($M_{conceptSim}$);
return $F_c$
```

Algorithm 5: Comparing Features

4.2.8 Similarity Function

The proposed idea is based on the calculation of features for each property of an concept. These features are compared pairwise using a similarity measure. The features are heterogeneous, consequently, the choice of an appropriate measure is not easy; the vectors include numeric values (e.g. average length), string values (like names) and sets of numbers or strings. For comparing strings, measure like the edit distance can be used. Numbers can easily be compared by calculating the absolute result of their difference. Both distances can be extended such that sets of numbers or strings are measured. The first measure we propose to compare heterogeneous feature vectors is called HFD (heterogeneous feature distance) and it is similar to the Euclidean function. It is defined as follows:

$$HFD(f, g) = \sqrt{\sum_{p=1}^{m} d_p(f_p, g_p)^2} \tag{4.3}$$

The feature vectors f and g are compared dimension by dimension (from 1 to m). d_p is a distance function that is specific for any kind of value pair the feature vectors can contain and it is defined as:

$$d_p = \begin{cases} ed(f_p, g_p) & \text{if } f_p, g_p \text{ are strings} \\ n_{\mathit{diff}}(f_p, g_p) & \text{if } f_p, g_p \text{ are numbers} \\ HFD(f_p, g_p) & \text{if } f_p, g_p \text{ are sets} \end{cases} \tag{4.4}$$

String pairs are compared using an implementation of the edit distance ed as introduced in Subsection 2.5. Numeric values are compared using the n_{diff} measure (explained below).

$$n_{diff}(f_p, g_p) = \frac{|f_p - g_p|}{range} \tag{4.5}$$

The range is used to normalize the difference of the two numbers and is calculated by considering all available values for this feature.
Sets of strings or numbers are recursively compared with the HFD measure.
f_p and g_p can not have different types, because the values of the vectors f and g are ordered (according to the different features). Furthermore, f and g are only compared, if they have the same type, too, i.e. the composition of the vectors is always the same.
The HFD measure is a normalized distance measure, and since the use of similarities is more usual in matching scenarios, it can be easily transformed into a similarity measure HFS by using the following function:

$$HFS(f, g) = 1 - HFD(f, g) \tag{4.6}$$

The HFD measure can easily be modified by using other distance functions for the different datatypes. The edit distance could be replaced by another string comparison measure like string equality or the n-gram measure. The similarity of numbers could be calculated in an unnormalized way or using a number equality measure.
There are also more possibilities for comparing sets of numbers, e.g. the cosine similarity. Additionally, instead of using the Euclidean function one can use the Manhattan distance

or another comparison measure (see Subsection 2.5 or [WM97] for an overview of different measures). For evaluation purposes we use some of these measures (see Section 7.1.3).

4.2.9 Propagating Similarity

Each property of a concept c_k is compared to each property of another concept c_m, i.e. the corresponding feature vectors are compared as described in the previous section. The result is a similarity matrix. All property pairs that have a similarity above a fixed threshold and that do not participate in another correspondences are part of the alignment (also see Subsection 4.2.10). Since the mapping should also contain pairs of matched concepts, the property similarity has to be propagated up to the concepts.

In our approach we concentrate on the following two propagation strategies:

- maximum approach
- stable approach

The *maximum approach* (see e.g. [EM07]) determines the maximum similarity for each property to another one, sums up all similarities and divides the sum by the number of properties. The advantage of this approach is the fact that all similarities and all attributes are taken into account. An illustration of the process is shown in the following example:

Assume having the two concepts (with their related attributes in brackets) *professor(name, surname)* and *prof.(name, family name, phone, eMail address).* The different attributes could have the similarities shown in Table 4.1.

sim	name	family name	phone	eMail address
name	0.9	0.3	0.1	0.1
surname	0.3	0.86	0.1	0.1

Table 4.1: Example attribute similarities

The maximum approach propagates the similarity from the attributes similarities to a concept similarity using the following equation:

$$sim_{concept} = \frac{\max_{sim}(name) + ... + \max_{sim}(eMailaddress)}{\#professor + \#prof.} \tag{4.7}$$

Thus, the maximum similarity value of each property is determined and the similarities of all properties are summed up. Following, this sum is divided by the number of properties. In this case, the concept similarity between “professor” and “prof.” is computed as follows:

$$sim_{concept} = \frac{1}{6} * (\max_{sim}(0.9, 0.3) + \max_{sim}(0.3, 0.86) + ...$$
$$... + \max_{sim}(0.3, 0.86, 0.1, 0.1)) \tag{4.8}$$

Consequentely, the similarity of the two concepts is determined by using the maximum approach is 0.62.

The *stable approach* is related to the “stable marriage problem”. We try to find a mapping between the two property sets based on the calculated similarities which is as optimal as possible, which has the consequence, that an entity is not always mapped to its maximum similarity partner. A detailed description of the used method, the Gale-Shapley algorithm, is given in [GS62]. The similarities of the mapped property pairs are summed up and normalized. An advantage of this approach is that it is more precise in respect to the definition of the matching problem, i.e. if concepts are similar there are very likely pairs of similar attributes. A disadvantage is that not all attributes participate in the mapping if the number of attributes of the two concepts differs and hence they are not considered when calculating the concept similarity.

To illustrate the stable approach we use the same concepts and similarities as in the previous example. The algorithm searches for a stable mapping configuration, the details of the Gale-Shapley algorithm are omitted here. In this case, the mapped property pairs are: (name, name) and (surname, familiy name). These attributes are mapped because they share the highest similarity (which is the most trivial case for a stable mapping). The similarity values of the mapped pairs are summed up and divided by their number. Hence, the concept similarity of the two concepts using the stable approach is 0.88.

4.2.10 Determination of a Mapping

The similarity propagation step transforms property similarities to concept similarities. Finally, we have to calculate a set of mapped concepts to extend the set of correspondences. In our case we determine pairs of concept that have a similarity above a fixed threshold t. All similarities below the threshold are rejected. To obtain the best mapping configuration out of the remaining concept pairs, the following strategy is used:

1. Search the concept pair (c_k, c_m) with the highest similarity value and mark it as mapped.

2. Reject all unmarked pairs (c_k, x) and (y, c_m) with x, y being any concepts.

3. Repeat step 1 and 2 until the list of mapping candidates is empty or only contains marked pairs.

This approach is a composition of the propagation strategies described in the previous subsection. In our evaluation the described strategy is used, but the maximum or the stable approach could also be used to determine a mapping.

4.3 Summary

In this chapter we presented two novel instance-based matchers. The first one uses regular expressions or catchwords to characterize attributes. For each concept, the regular expressions describing its attributes are represented in a RECW vector. These vectors are compared using the cosine measure to find concept correspondences. The concept similarity can be propagated downwards to the property level.

The second approach uses the instances of an attribute to calculate several features depending on the type of the property. These features can be compared pairwise using a similarity function. The property similarities can then be used to find concept correspondences.

Instance-based methods are rather time-consuming and not all concepts or properties contain instances. Consequently, the instance-based matchers should be integrated into a complex matching system, which also contains schema and structure matcher. In the next chapter `MICU` is introduced, which is a matching system that combines different matchers and interacts with the user.

5 The Matching System

The instance-based matchers described in the last section are only applicable for concepts with instances and in general, not all concepts of an ontology contain instances.

In most hierarchical ontologies instances only exist for leaf concepts, e.g. ontologies describing web directories include the different categories and the relations among them, but instances (documents or links), can only be found on the lowest level. Hence, it is very useful to combine instance-based matchers with other matching methods exploiting schema and structure information. In the scope of this thesis we integrated the two instance-based matchers into a complex matching system name `MICU` (Matching using Instances, Concepts and the User), which makes use of previous alignments and several schema- and structure-based matching methods. Additionally, the user interacts with the matching system during the matching process and helps increasing the matching quality.
The remainder of this chapter is organized as follows: Section 5.1 presents the architecture of `MICU`. The used matching methods are explained detailed in Section 5.2. The user interaction and the reuse of previous alignments is described in Section 5.3. `MICU` is evaluated using `ONTOBI` and the OAEI benchmark; the results are described and discussed in the separate evaluation Chapter 7.

5.1 Architecture

`MICU` is a matching system that combines concept-, instance- and structure-based matching methods and interacts with the user to increase the match quality. The architecture is shown in Figure 5.1, summarized in Algorithm 6 and explained afterwards.
Parsing and Extracting

First of all, the ontologies are parsed and all pieces of meta information like concept and attribute names or comments are extracted. Additionally, for each concept it is noted if there are instances belonging to it, but the instances are not extracted, yet.

```
input  : ontologies $O1, O2$
output: alignment $A$
Parsing and Extracting($O1, O2$) $\rightarrow$
global set $E = \{(e_i, e_j), e_i \in O1, e_j \in O2\}$
$A = A \cup ReuseOfAlignment(E)$;
$A = A \cup SchemaBasedMethods(E)$;
$A = A \cup InstanceBasedMethods(E)$;
$A = A \cup StructureBasedMethods(E)$;
$A \leftarrow AskUser(E)$;
return $A$
```

Algorithm 6: Matching Process

Reuse of alignments

To reuse alignments of previous match results, all mappings that have been confirmed by a user in previous matching processes are stored in a database. This database is queried for each concept, attribute or relation pair as the first step in the matching process. If the pair exists in the database, the corresponding similarity value is obtained. If this value is above a certain threshold t_{mapRep}, the concept pair is directly added to the alignment. The advantage of this procedure is that the matching process of entities that already have been matched in previous matching processes can be shortened.

Schema-based methods

After looking up in the mapping repository, the remaining entity pairs are compared using several schema-based methods, which exploit concept, attribute and relation labels and comments. Pairs that have a similarity above a certain threshold t_{sch} are directly added to the alignment; pairs with a similarity above a threshold $t_{mapCand}$ but beneath t_{sch} are presented to the user who has to decide, whether the entities have to be aligned or not. Confirmed pairs are added to the alignment, rejected pairs are still considered in further matching methods.

Instance-based methods

All remaining pairs in which both participating entities provide instances are the input of the instance-based matching methods. This process is mainly performed on the attribute level, because our instance-based matchers use attribute instances, but the similarities of the attributes are used to calculate concept correspondences as well. Again, pairs having a similarity above a threshold t_{inst} are added to the alignment, and all pairs with a similarity between $t_{mapCand}$ and t_{inst} are proposed to the user who has to reject false correspondences.

Structure-based methods

Finally, the remaining entities are used to perform a structure-matching; several constraints are used to determine additional correspondences based on already matched entities.

The user is again demanded to reject false correspondences and finally, an overview of the whole alignment is shown. The user gets another possibility to correct the alignment before it is stored.

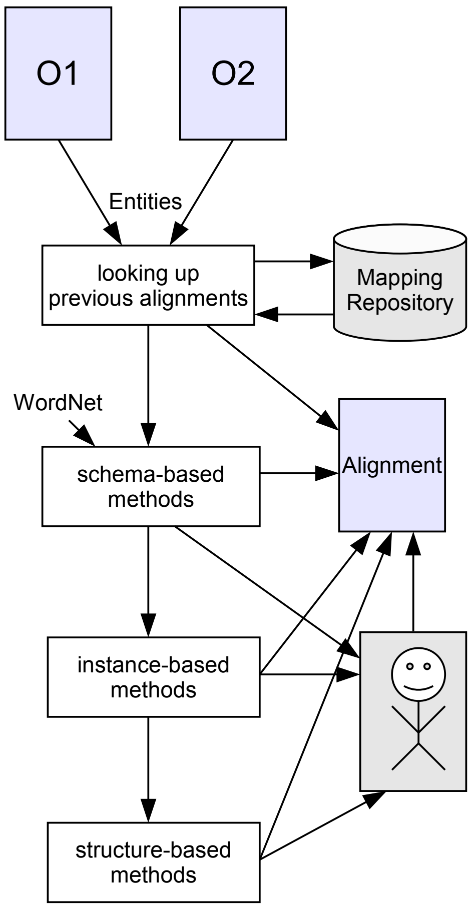


Figure 5.1: Architecture of MICU

The whole matching process is designed to reduce the time complexity and to benefit from the interaction with the user. The set of entities that participate in the different matching methods is reduced in each step. First of all, already matched entities are detected while scanning the alignment database, such that their similarity does not need to be recalculated again. The actual matching process starts with schema-based methods, because very similar labels or comments are a strong hint for a correct correspondence. The user only has to look at correspondences, that have a certain similarity, and he is demanded to reject false correspondences. Due to this process, the entity set that takes part in the time-consuming instance-based matching is reduced. Additionally, only concepts or attributes providing instances are considered at all. After performing the instance-based matching, the user is demanded to reject false correspondences again. The set of correct correspondences, that have been found during the matching process by now, provides a good basis for proceeding with the matching process using the structural information of the ontologies. Already matched entities work like anchors from which the similarity is propagated to unmatched nodes

using several constraints.
It might seem like the user has to interact with the system very often, but each time, when he is asked to reject mappings, he only has to examine a small set of correspondences (compared to the whole set of entities) and his decisions enhance the quality of further matching process. So, in total the user has no more work to do than in traditional user feedback dialogs that appear at the end of the matching process, but in most cases the effort is reduced.
The full automation of the matching process does not seem to be possible without decreasing the quality of the matching result.

5.2 Used Methods

In the description of `MICU` several matching methods have been mentioned. In the following the different methods are presented in detail and they are labeled using the classification described in Section 2.6.

5.2.1 Schema-Based Methods

The schema-based matching methods used in `MICU` mainly use concept, attribute and relation labels. Additionally, comments are used and WordNet is queried to obtain synonyms and to obtain comments, if not provided by the input ontologies.
With respect to the classification given in Section 2.6, the schema-based methods are mainly syntactic (on the element level) and external (on the element level).
Three methods are applied on entity labels: string similarity based on the edit distance or on the n-gram measure and a prefix-/suffix-similarity. The string similarity based on the edit distance is shown in Equation 2.6; the string similarity using the n-gram measure is stated in Equation 2.4 with n set to 2.
The prefix-/suffix similarity of two strings s and t is defined as follows:

$$sim_{preSuf}(s,t) = \begin{cases} 1 \text{ if } s = t_1 t \lor s = t t_2 \lor t = s_1 s \lor t = s s_2 \\ 0 \text{ else} \end{cases}, \tag{5.1}$$

for some t_1, t_2, s_1, s_2.
It tests whether one entity label is the prefix or the suffix of the other label. In the case, that the value of this similarity measure is 0 it is completely deleted out of the similarity calculation process, because the fact, that the labels do not have a prefix or suffix similarity, has no significance for the real similarity of two labels.
For each entity label WordNet is queried to obtain its synonyms. The process could be accelerated by building up a synonym database which is filled and queried during the matching process, such that the synonym information can be reused in future matching processes. For each label there might be a set of synonyms, which are compared using Equation 2.10. If only one or none of the labels has synonyms, the similarity between the synonym sets is 0. For optimizing reasons, we do not consider the synonym measure in this case, but we exclude

it from the similarity calculation. The fact, that labels do not have synonyms should not have an influence on the similarity; otherwise two equal labels, that do not have a synonym, would have a synonym similarity of 0, which would decrease their overall similarity.
Most ontologies include comments that describe entities in terms of natural language. These comments can be used to determine the semantics of the entities. If comments are not available, `MICU` searches for definitions in WordNet (this process can again be accelerated by storing extracted information in a database to reuse it in further matching processes) and treats them as comments. For each pair, that provides comments for both entities, the comments are stemmed and stop words are eliminated. After that, the normalized comment sets are compared using Equation 2.10. Similar to the handling of labels that do not have synonyms the absence of comments should not influence the similarity in a decreasing manner. Hence, the comment measure is not considered if comments are not available.
The similarities of the methods described before are aggregated using Equation 2.13.
Each entity pair with a similarity above t_{sch} is directly added to the set of new correspondences A_1; pairs with a similarity between $t_{mapCand}$ and t_{sch} are shown to the user, who decides if these pairs are also added to A_1 or if they stay in the entity set E. Finally A_1 is returned. The whole process is again described in Algorithm 7.

5.2.2 Instance-Based Methods

The instance based methods used in `MICU` are the ones described before in Section 4.1 and Section 4.2. The feature approach is always executed if instances are available. The number of instances used for feature calculation has to be determined by the user before starting the matching process. The process as described in Algorithm 5 in Subsection 4.2.7 is stopped after calculating the similarity matrix M_{sim}, and the attribute correspondences are obtained. Subsequently, the similarities are used to propagate concept correspondences within the SimilarityPropagation step, which creates a concept similarity matrix $M_{conceptSim}$. This matrix provides the basis of determining concept correspondences.

The instance-based matching approach using regular expressions is not used by default, because it demands further effort done by the user, i.e. if available, an appropriate file has to be selected or regular expressions have to be specified. If wished by the user, the approach is executed as described in Figure 4.2.
The whole process is presented in Algorithm 8.

If the regular expression approach is not executed, the similarity of an entity pair is defined as $sim_{o,p} = \text{FeatureApproach}(I_o, I_p)$.

```
input  : entity set $E$
output: alignment set $A_1$
similarity matix $sim_{schema}$;
forall $(e_i, e_j) \in E$ do
    $sim_{i,j}$ =StringSimilarityED($e_i, e_j$);
    $sim_{i,j} = sim_{i,j}$ + StringSimilarityNGram($e_i, e_j$);
    define $methodCounter := 2$;
    if $e_i, e_j$ have synonyms (included or extracted from WordNet) then
        $sim_{i,j} = sim_{i,j}$ + SynonymSet($e_i, e_j$);
        $methodCounter + +$;
    end
    if $e_i, e_j$ have comments (included or extracted from WordNet) then
        $sim_{i,j} = sim_{i,j}$ + CommentSet($e_i, e_j$);
        $methodCounter + +$;
    end
    $sim_{preSuf}$ =PrefixSuffixSimarity($e_i, e_j$);
    if $sim_{PreSuf} == 1$ then
        $sim_{i,j} = sim_{i,j} + sim_{PreSuf}$;
        $methodCounter + +$;
    end
    $sim_{schema} \leftarrow sim_{i,j}$;
end
correspondence set $A_1$;
user set $U_1$;
forall $sim_{k,l} \in sim_{schema}$ do
    if $sim_{k,l} \geq t_{sch}$ then
        $A_1 \leftarrow (e_k, e_l)$;
    end
    if $t_{mapCand}$ ¡ $sim_{k,l} < t_{sch}$ then
        $U_1 \leftarrow (e_k, e_l)$;
    end
end
$A_1 = A_1 \cup AskUser(U_1)$;
$E = E - A_1$;
return $A_1$
```

Algorithm 7: SchemaBasedMethods

```
input  : entity set $E$
output: alignment set $A_2$
forall $(e_o, e_p) \in E; I_o \neq \emptyset, I_p \neq \emptyset$ do
    $sim_{o,p}$ = FeatureApproach($I_o, I_p$) or RegExApproach($I_o, I_p$);
    if $sim_{o,p} \geq t_{inst}$ then
        $A_2 \leftarrow (e_o, e_p)$;
    end
    if $t_{mapCand} < sim_{o,p} < t_{inst}$ then
        $U_2 \leftarrow (e_o, e_p)$;
    end
end
$A_2 = A_2 \cup AskUser(U_2)$;
$E = E - A_2$;
return $A_2$
```

Algorithm 8: InstanceBasedMethods

5.2.3 Structure-Based Methods

The structure-based method used in `MICU` is executed as a final step to refine the mapping. It uses the correspondences and similarities determined by the schema- and instance-based methods and propagates the similarity to adjacent nodes. For this purpose an algorithm that is similar to the Similarity Flooding algorithm [MGMR02], which will be explained in the following.
Similarity Flooding is a graph matching algorithm that has been adapted to schema matching. For this purpose, schemas are transformed into directed graphs. An initial mapping between all nodes of these graphs is determined using a `StringMatch` operator, which compares common prefixes and suffixes. The initial mapping is then used as the starting point for the main algorithm, which produces a refined mapping using a fixpoint computation.
First, a similarity propagation graph is constructed, in which every node represents a candidate mapping pair of the two input schemas. The edges represent the original relations between the entities in their source schemas and additionally an extra edge is inserted in the reverse direction (because a similarity can be propagated upwards and downwards). To every edge a weight, that indicates how well a similarity propagates to the adjacent nodes, is assigned. The weight for each edge depends on the number of outgoing edges of the corresponding node, i.e. if a node has two outgoing edges, the weight assigned to these edges is 0.5, for three edges it is $\frac{1}{3}$ and so on.
Using this similarity propagation graph, the similarities of the initial mapping are propagated, i.e. flooded, through the graph by iteratively multiplying the similarities with the edge weights and summing up the values for all incoming edges of a node. The process is iterated until a fixpoint is reached or until a fixed number of iterations is executed.
Finally, a filter, like a threshold filter, is used to determine the final mapping.

In MICU we use the similarity flooding algorithm as described above, only the initial mapping is not obtained using the StringMatcher but using the results of the schema-and instance-based matchers executed before. Furthermore, the similarities are only propagated to unmatched nodes, i.e. the similarities of already matched nodes are not changed and we only perform one iteration. As a final filter we use a simple threshold t_{struct}.
According to the classification used in Section 2.6, this matcher is a syntactic structure-level matcher.
The whole structure-based matching algorithm is presented in Algorithm 9.

```
input  : entity set $E$
output: alignment set $A_3$
matrix $sim_{struct} \leftarrow$ Structure-Based Matching($E$)
user set $U_3$:
forall $sim_{o,p} \in sim_{struct}$ do
    if $sim_{o,p} \geq t_{struct}$ then
        $A_3 \leftarrow (e_o, e_p)$;
    end
    if $t_{mapCand} < sim_{o,p} < t_{struct}$ then
        $U_3 \leftarrow (e_o, e_p)$;
    end
end
$A_3 = A_3 \cup AskUser(U_3)$;
$E = E - A_3$;
return $A_3$
```

Algorithm 9: StructureBasedMethods

5.3 Alignment Reuse and User Feedback

The reuse of alignments is a good way to reduce the time complexity of the matching process. All correspondences, that are included in a final alignment of a matching process, are inserted into a database together with their similarity values. Further matching processes can then reuse the correspondences included in the database to find first matching correspondences. Another advantage besides the saving of time is that entities might not have instances in the current matching process but in previous ones. If the schema information of such entities is dissimilar, we can find this mapping correspondence anyway by using the previous alignments. It is obvious, that entities aligned in previous matching processes do not necessarily have to be matched in the current process, but this is a uncertainty with which we have to cope in all other matching methods, too.
The process of reusing previous alignments is shortly described in Algorithm 10.

As explained in Section 5.1, in most cases it is necessary to interact with the user to increase the matching quality. The goal of MICU is to find a balance between user action

```
input  : entity set $E$
output: alignment set $A_0$
forall $(e_i, e_j) \in E$ do
    query database $D$;
    if $(e_i, e_j) \in D and sim(e_i, e_j) > t_{mapRep}$ then
        $A_0 \leftarrow (e_i, e_j)$;
    end
    $E = E - A_0$;
end
return $A_0$
```

Algorithm 10: ReuseOfAlignment

and automated process (as proposed in [Hea09]). The user should not be overburdened, but the quality should be increased. Consequently, we decided to ask the user for rejecting false correspondences from time to time to make advances of the short concentration phases of humans.

```
input  : global entity set $E$; set of candidate correspondences $U$
output: alignment set $A_U$;
forall $(e_i, e_j) \in U$ do
    ask user to reject;
    if $(e_i, e_j)$ is not rejected then
        $A_U = A_U \cup (e_i, e_j)$;
    end
end
return $A_U$
```

Algorithm 11: AskUser

5.4 Summary

In this chapter we presented a novel matching system called `MICU`, which unifies the advantages of different kinds of matching methods. Additionally, previous alignments are reused, WordNet is queried and the user is involved in the matching process. The architecture of `MICU` is flexible, such that new methods can be included at any time.
`MICU` as well as the two novel instance-matchers described before need to be evaluated. For this purpose, we present ONTOBI in the next chapter. ONTOBI is a benchmark, which contains many classes and properties, and especially a huge amount of instances.

6 ONTOBI - An Evaluation Benchmark with many Instances

Matching systems or single matching methods need to be tested to determine their quality for different scenarios. In some cases it is important that a matching strategy works well for a special application, in other cases (if the application space is not limited) methods should be all-rounders. In each case, an extensive evaluation is advisable to detect weak and strong points of a method or system and to determine an overall quality. Most evaluation frameworks focus on the evaluation of the matching quality and disregard the time complexity.
In general there are different types of evaluation (see Subsection 2.7). The aim of this thesis is to develop matching methods and a matching system that work as good as possible and solve many different kinds of heterogeneities. For testing purposes we decided to use a benchmark test series because the variety of the test cases is very huge and it is possible to determine weak and strong points very detailed.
There are already a few benchmarks available as described in Subsection 3.2 but they are mostly not usable for testing instance-based matching methods because the number of available instances is very small and there are either no variances on the concept-level or on the instance-level. Hence, we decided to build our own benchmark which includes heterogeneities on the concept, structure and instance level and which provides a huge amount of instances. Our first approach has been described in [Zai08a] and [ZCV10]; a user can dynamically build his own ontology by browsing Wikipedia and selecting the information that should be stored in the ontology. This process works semi-automatically but it is very time-consuming and includes some sources of errors. Consequently, we decided to use a part of DBpedia [LBK$^+$09] which is an ontology based on Wikipedia. In this way we obtain a reference ontology that provides the basis for our benchmark. A set of transformations is applied on this reference ontology such that we obtain heterogeneous ontologies which have to be matched against the reference ontology. For applying the transformations, an ontology modificator has been

developed which allows to execute one or more transformations on any ontology, outputs the new ontology in the same format as the input ontology and also creates the reference alignment on the fly represented as described in the Ontology Alignment API [Euz06]. For the benchmark proposed in this thesis, ONTOBI, we apply a well-defined set of transformations and create 17 test cases, each testing different types and grade of heterogeneities.
The remainder of this chapter is organized as follows:
The first section describes the construction of the reference ontology: the construction using Wikipedia is described in Subsection 6.1.1 and the use of DBpedia is presented in Subsection 6.1.2. An overview of the transformations is given in Subsection 6.2 and the process of creating test cases is explained in Subsection 6.3. An evaluation concerning the predefined requirements is given in Subsection 6.4. The chapter concludes with a summary in Subsection 6.5.

6.1 The Reference Ontology

The requirements mentioned in Section provide a basis for the development of our benchmark named ONTOBI (ONTOlogy Matching Benchmark with many Instances). Additionally, we formulated more precise criteria that need to be fulfilled to guarantee the compliance with these requirements. They are derived from our experience with existing matching systems/algorithms and from the types of heterogeneity described in Subsection 2.4:

- large ontologies: most instance-based matchers need a certain number of instances to achieve best quality. Furthermore, the time complexity is an interesting issue when evaluating instance-based approaches. The instances of this benchmark should be realistic (not artificially created) and the variety of different values should be high (if the number of values is not limited by definition). This criteria is very important to reach equal conditions for all kinds of matching systems.

- differences in structure: similar to the OAEI the structure of the ontologies should be flattened or expanded in some cases, i.e. the ontologies should be conceptually heterogeneous. This is very important since a lot of matchers exploit the structural information of an ontology and structural heterogeneity is an important issue for all matching systems.

- differences in instances: instances that are semantically equal can be formated in different ways, one good example is the date. Matching systems should be able to detect semantical similarity despite of structural differences. Additionally, instances can include spelling mistakes or the instance sets can be completely different.

- differences in schema: given that the ontologies should be as realistic as possible schema variances like spelling mistakes, abbreviations or synonyms should be implemented as well.

The benchmark described in Subsection 3.2 and ONTOBI will be compared according to all these requirements later on in Subsection 6.4.

First of all, a reference ontology has to be created, which has to fulfill some more properties. Obviously, the number of concepts has to be high enough to express different structures (different properties, relations, different conceptualizations to find 1:n mappings etc.).

The properties of a concept are either data type or object properties and both possibilities should be represented. Optionally, the ontology could contain different concepts having attributes expressing the same real world thing but modeled by different properties (e.g. a date can be modeled as data type or object property).

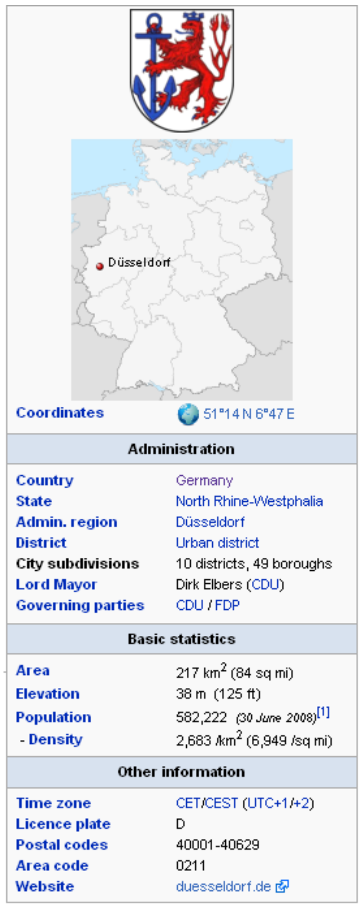
In particular, this benchmark should be usable for instance-based matchers, i.e. the ontology should contain an appropriate number of instances as well. To build a realistic benchmark instances should not be created artificially.

The data types of the instances should be varying, i.e. not all information should be stored in strings.

Since a lot of matching systems use comments to produce an alignment, most of the concepts in the reference ontology should contain comments, too. Preferably all meta information and the instances should be translatable into another language.

6.1.1 Dynamically Exploiting Wikipedia Infoboxes

Our first approach for obtaining a reference ontology semi-automatically exploits Wikipedia to obtain concepts, attributes, relations and instances. For this purpose it makes use of the info boxes provided for many articles, because they represent knowledge in a structured form such that it can easily be used to model an ontology. An example info box is shown in Figure 6.1. Each info box displays a set of attributes of a concept together with values for a special instance of this concept. As one can see in the example an info box mainly consists of two columns. The first column includes the attribute names, e.g. country and state. These attributes are assigned to a concept which gets a user-defined name in our approach, in this case e.g. “city”. The values in the second column represent attribute instances, which are linked to further concepts. The links are automatically extracted and represent object properties. So, for each concept that should be included in the ontology an exemplary info box is searched and the corresponding attributes are extracted and assigned to the concept. Relations between concept are created automatically, but it is possible to add user-defined relations as well. After finishing the process of creating concepts and relations, instances can be extracted. Instances can be chosen manually or a complete list of instances can be extracted at once. The assignment of instances to concepts works automatically.
The evaluation of this approach showed, that it generally works well but in some cases not all values are extracted out of a info box and the automatic assignment of the instances is error-prone. Further studies showed that there is already an ontology representing the knowledge of Wikipedia called DBpedia.



Coordinates	51°14 N 6°47 E
Administration	
Country	Germany
State	North Rhine-Westphalia
Admin. region	Düsseldorf
District	Urban district
City subdivisions	10 districts, 49 boroughs
Lord Mayor	Dirk Elbers (CDU)
Governing parties	CDU / FDP
Basic statistics	
Area	217 km^2 (84 sq mi)
Elevation	38 m (125 ft)
Population	582,222 *(30 June 2008)*[1]
- Density	2,683 /km^2 (6,949 /sq mi)
Other information	
Time zone	CET/CEST (UTC+1/+2)
Licence plate	D
Postal codes	40001-40629
Area code	0211
Website	duesseldorf.de

Figure 6.1: Info box for “Duesseldorf”

6.1.2 Using DBpedia

DBpedia is a project that has the goal to provide the knowledge stored in Wikipedia in a web-accessible knowledge base that can answer complex queries (e.g. list all films of Tom Cruise, in which Cameron Diaz also played a role). The ontology provided in this project includes the most commonly used info boxes of Wikipedia and a huge amount of instances as well and can be queried using different interfaces that are e.g. based on SPARQL [EP06]. The concepts of this ontology describe persons, places, organizations, species, buildings and works. For constructing our reference ontology we choose version number 3.4, which contains 205 classes, 1144 object properties and 1024 data types properties. The DBpedia ontology only contains rdfs:subClassOf and owl:ObjectProperty relations and is strictly hierarchical.

The most important subject when creating the reference ontology for ONTOBI is providing a big instance set, because this is an issue disregarded in other approaches. We decided not to use all instances because the set is too huge (over one million instances) and we wanted to create an ontology with a varying range of instances. Finally, we extracted 17 to 576 instances for more than 50 percent of the concepts on the lowest ontology level; that makes a total amount of 13704 instances at the moment. During the process of including instances, we added further 8 classes, 37 object properties and 87 data type properties.

The ontology is written in terms of OWL-DL and presented in RDF/XML format (similar to the ontologies created for [OAE09]) and the meta information and instances are written in English.

The DBpedia ontology does not provide comments in its schema ontology. Since these are used by many matching algorithms, comments describing the concepts are added using their definitions in WordNet, if available.

6.2 Transformation of Meta Information and Instances

Once we obtained a reference ontology, we transform it to create a set of modified ontologies. These ontologies are mapped against the reference ontology. According to the different types of heterogeneity described in Section 2.4 and the requirements explained in Section 2.7 and Subsection 6.1, several transformation rules are defined; an overview is given in Table 6.1. The modifications can be divided into simple and complex ones; the partitioning is mainly based on experience with different matching systems and the development of an own matcher respectively. Simple modification rules can be divided into two parts:

1. they only pertain one part of an ontology, i.e. either the meta information, the structure, or the instance set.

2. one type of matching algorithm can cope with the effect of the modification, e.g. spelling mistakes can be "corrected" by using simple measures like the edit distance.

Simple modifications usually should not influence the matching quality significantly. Complex modifications are much more difficult to handle and have effect on various elements of the ontology.

In the following the different transformations are explained in detail:

- M describes a set of operations resulting in spelling mistakes, i.e. insertion/deletion of one character or switching of two characters. It is applied to concept and property names or instance values as well and the executed operation is determined randomly in each case.

- Instance values may be formatted in different ways. A good example is the date: dates may be expressed as object properties related to a concept "date" or as a data type property. In the latter case the RDF data type of the "date" may be string or date, and the format may be "dd.mm.yy" or any other. The F transformation aims to randomly vary the formats for a certain amount of instance values.

- Concepts may include comments (rdfs:comment property), that describe the semantics of this concept. This information can be used to detect homonyms or synonyms. By applying transformation S1 for all entities the comments are fully deleted.

identifier	modification	applied on
simple transformations		
M	spelling mistakes	$C, A(c_k), I(c_k)\ \forall c_k \in C$
F	changed format	$I(c_k)\ \forall c_k \in C$
S1	suppressed comments	$C, A(c_K)\ \forall c_k \in C$
S2	no data types	$I(c_k)\ \forall c_k \in C$
I1	overlapping data sets	$C_o \subseteq C$
complex transformations		
H1	expanded structure	
H2	flattened structure	
L1	another language	$C, A(c_k)\ \forall c_k \in C$
L2	random names	$C, A(c_k)\ \forall c_k \in C$
L3	synonyms	$C, A(c_k)\ \forall c_k \in C$
I2	disjunct data sets	$C_o \subseteq C$

Table 6.1: Overview of the modifications

- Modification S2 has the effect that all rdfs:range properties assigning data type properties to data types and object properties to resources are deleted. Additionally, instance values (i.e. literal objects) can be assigned to XML data types, e.g. http://www.w3.org/2001/XMLSchema#date. This assignment provides additional information on the semantics of the instance values and the corresponding concept. The application of S2 results in the repression of all data type assignments.

- As a default case, the instance sets of the ontologies are identical. Modification I1 replaces a part of the instance values by new ones. Depending on the number of new instances, this modification is simple or more complex. The exchanged instances are distributed randomly according to the concepts they belong to.

- The structure of the ontology is another important hint for most matching algorithms. The application of H1 results in an expanded hierarchy, i.e. several top-level concepts are added to the ontology and the structure might be rearranged as well.

- H2 is the opposite operation to H1; the hierarchy structure is flattened by deleting (all) top-level concepts, such that all concepts are on the same level.

- Meta information and instances may be written in different languages. The application of L1 causes the translation of all concept and attribute names and all comments to another language.

- The meta information of an ontology might be completely senseless due to human or technical reasons. L2 simulates this case by replacing all concept and/or attribute names by random strings.
- The meta information of ontologies is always created subjectively, hence the same real world objects are often described with different labels. Transformation L3 replaces concept and attribute names by synonyms (if possible).
- Modification I2 results in an exchange of the whole instance set. The distribution of the new instances according to the concepts is chosen randomly.

6.3 Creating Test Cases

The different test cases shall provide a basis for testing different parts of a matching system and the system in general. Therefore, different pieces of information are repressed or changed gradually by applying one or more of the predefined operations. Each test case consists of two ontologies to be matched: the reference ontology and one modified reference ontology. To simplify the evaluation of the matching results, the correct reference alignment is given. The format for this alignment is borrowed from [Euz06] (Ontology Alignment API). An overview of the process is given in Figure 6.2. As evaluation measures, the standard measures precision and recall, and their combination, the F-measure, are proposed.

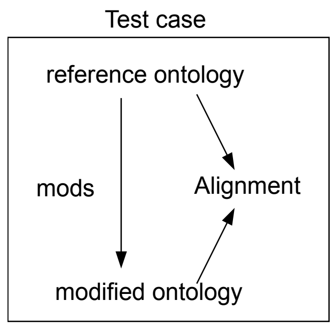


Figure 6.2: Overview of a test case

6.3.1 The Ontology Modificator

In addition to the development of a complete benchmark test series we want to provide a possibility to create test cases dynamically and for various ontologies. For this purpose we constructed and implemented an ontology modificator that gets an ontology as input, applies one or more of the transformations described above and outputs the modified ontology. Furthermore, the reference alignment should be produced such that it can be used for further evaluation of self-produced matching results between input and output ontology. The modificator provides the following possibilities:

- insertion of (additional) comments using WordNet
- creation of misspellings (concept/property labels and/or instance values)
- repression of data type assignments and/or labels
- replacement of concept and property labels by random strings
- substitution of concept or property labels with synonyms using WordNet (if possible)
- changing of instance format
- flattening of hierarchy
- distribution of instances
- creation of reference alignment

Until now is not possibly to extend the hierarchy automatically, because a user has to specify the name of superconcepts manually, such that it makes sense. Additionally it is not possible to create new instances; the user has to provide a set of instances which can be distributed over the two ontologies depending on the chosen transformation I1 or I2 (or disjunct datasets). The modificator copies the input ontology and changes the statements that are pertained of the selected transformation. If instances should be included in the output ontologies the corresponding statements are added and finally the transformed, heterogeneous ontology is generated. Additionally, the reference alignment is written using the format described in [Euz06].

6.3.2 Test Cases

Using the reference ontology as the starting point a systematic benchmark is created. For this purpose the reference ontology is modified using one or more of the transformations mentioned in Section 6.1. Most of the transformations are done automatically by using our ontology modificator which has been described in the previous subsection. The modification H1 (extension of structure) is done by hand using the Protégé [Pro09] ontology editor to guarantee the quality of the ontologies. The translation to another language, test L1, is not created yet, because due to the high number of classes and properties this process is very time-consuming.
In general, there are a lot of different possibilities to combine the transformations. For testing purposes we choose a set of possibilities according to our personal experience with matching systems, i.e. we choose those cases which we think are most interesting or most challenging. Additional test cases can be created at any time using the ontology modificator.
The combination of the different modifications gradually increases the difference between the modified and the reference ontology, such that the matching task gets more complex.

test number	modification(s)
simple tests	
OS1	M
OS2	S1
OS3	I2
OS4	L1
OS5	L2
OS6	L3
OS7	H1
OS8	H2
complex tests (two mods)	
OC1	M, S1
OC2	L2, S2
OC3	L3, I1
OC4	H2, I1
complex tests (at least three mods)	
OCC1	M, S1, S2, I2
OCC2	M, L3, S2
OCC3	L3, H1 , I2
OCC4	S1, F, I1

Table 6.2: Overview of the benchmark

An overview of the complete benchmark series can be found in Table 6.2. The ontologies in the simple tests series are reference ontologies, which have been modified by using one single transformation. These tests can be very helpful to determine the quality of a single matcher, which focuses on one part of an ontology. The complex tests are divided into two parts. The tests OC1-OC4 combine two transformations with the goal to create a more difficult test without deleting too much information. The combinations consist of two modifications that concern different parts of an ontology (structure, instance set or meta information of concepts and property) or that are not that strong when applied solely. At least one part of the ontology stays equal to the referring one of the reference ontology.

The tests OCC1-OCC4 are the most difficult ones and use three or more combinations. In these cases the loss of information is significant and to gain good matching results, the matching system must provide and combine various matchers.

In the following, the goals of the test cases are described more in detail:

- simple tests : These tests only use one modification to change the reference ontology, i.e. the matching task is quite simple for a complex matching system. But it might be reasonable to test single matching methods and for this purpose these test cases are appropriate. The instance set stays the same for all tests besides OS3. Language-based matching methods can be tested using OS1, OS2 and OS4-6, structural algorithms might be challenged in test case OS7 and OS8. The transformation F and S2 are too weak to be applied solely.

- complex tests (2 modifications):

  - OC1 combines M and S2 to create an ontology with reduced meta information. M is also applied to the instance set, such that the instance sets differ slightly, too. The structure stays the same, such that especially name-based or language-based techniques can be tested.

  - OC2 is a difficult test, because the concept and attributes are replaced by random strings and data types are repressed. Comments are still available, hence systems can capture the semantics of the concepts by applying language-based techniques and by analyzing instance values.

  - OC3 combines the use of synonyms with an exchange of some instances. The amount of overlapping instances is about 1000, and the instances are not distributed equally. This test is appropriate to evaluate a combination of language- and instance-based matchers or the use of structural matchers which are mainly not influenced by the modifications.

  - OC4 combines disjunct data sets with a flattening of the hierarchy. This task is quite easy for language-based matchers, since the meta information stays the same, but it is a challenge for instance-and structure-based methods, because there is no structure at all (all concepts are on the same level), and the instances sets are totally different.

- complex tests (at least 3 modifications):

  - OCC1 changes the meta information (misspelling and no comments) and the instance set (misspelling, no data types, disjunct data sets). This task is especially useful for instance matchers, which have to cope with disjoint, “false” and incomplete data (due to the misspellings and the lack of data type assignments).

  - OCC2 causes a significant change in the meta information of the ontology, because the concept and attribute names are replaced by synonyms, which include misspelling as well, and comments are repressed totally. Instance set and structure stay the same, such that this task is a challenge for all language- and name-based matchers.

- OCC3 transforms all parts of the ontology and is therefore appropriate to test a combination of different matchers. Names are replaced by their synonyms, the hierarchy is expanded and the structure of the ontology is rearranged; additionally the instance sets are disjoint.
- OCC4 concentrates on the modification of the instance set by changing the format of all date values (e.g. from yyyy-mm-dd to dd.mm.yyyy), repressing the data type assignments and generating overlapping instance sets.

Certainly, one can think of lots of other combinations and test cases, but we tried to focus on the most interesting and probably most useful ones. The test cases aim to challenge different kinds of matching methods, such that each test case might be more or less difficult for the different matchers (matching systems). Another important issue is the combination of different matchers. A system, that provides different kinds of matchers (name-/structure-/instance-based) has to find good combination strategies to obtain a good matching quality throughout all benchmark tests. For this purpose, we tried to vary the modified parts of the ontology and the difficulty of the different tests.
The matching of large ontologies is also an issue that needs some more studies. Most of the matching systems can not work with very large ontologies with a good performance (if they are able to load the ontologies at all). The ontologies of this benchmark are very large, such that it can be used for testing performance and scalability, too.

6.4 Fulfillment of Requirements

In Subsection 2.7 and Subsection 6.1 several requirements on evaluation benchmarks have been mentioned. Table 6.3 shows the quality of ONTOBI according to these requirements and coincidentally compares ONTOBI with the benchmarks described in Subsection 3.2.

First of all, we want to describe the result of ONTOBI in more detail: The previous sections clearly describe the systematic procedure of ONTOBI. The test cases are the same for all participants that want to use it. A reference alignment is given, such that the results are non-ambiguous. The continuity can be assured if the systems repeat the tests in certain time intervals. As evaluation rules the measures precision, recall and f-measure are proposed. As long as the benchmark is not used officially within a workshop or the like, the compliance with these rules can not be checked. i.e. we can not make an universally valid statement concerning the continuity. The quality of the ontologies extracted from DBpedia is quite good; the reliability on the data obtained from Wikipedia is given, because the instances are assigned to the correct concepts and for matching purposes it does not matter if the information is always up-to-date. The modifications are very manifold and the combinations try to balance the parts of the ontologies that are changed such that no type of matching system is favored or disadvantaged, i.e. the equity is ensured. This is also caused by the fact, that ONTOBI provides differences in all three parts of the ontology, i.e. in schema, structure and instance set. The benchmark provides a huge set of instances and the number

	OAEI Benchmark	A-R-S T-S-D	IIMB	STBenchmark	ONTOBI
systematic	+	+	+	+	+
continuity	+	+	+	n/a	n/a
quality	+	+	+	n/a	+
equity	o	-	-	n/a	+
large onto.	-	-	o	n/a	+
diff. in schema	+	-	-	o	+
diff. in structure	+	-	-	+	+
diff. in instances	-	+	+	o	+
dissemination	+	+	+	-	o
intelligibility	+	+	+	-	o

Table 6.3: Comparison of Evaluation Benchmarks: **+** fully satisfied, **o** partly satisfied, **-** not satisfied

of concepts and properties is also high. Dissemination and intelligibility are also ensured, since the benchmark is available publicly and the correct reference alignment is given (see http://dbs.cs.uni-duesseldorf.de/projekte/ONTOBI/).
The OAEI Benchmark fulfills most of the requirements, but one of the most important points, i.e. the equity can not be assured fully, because instance-based methods are disadvantaged due to the lack of a reasonable amount of instances. Furthermore, the ontologies are quite small and there are no modifications executed on the instance set (except the deletion of instances).
The IIMB only provides differences on the instance level, such that equity can not be assured (but to remember: IIMB was built to test instance matching methods, not instance-based matching methods). Additionally, the ontologies are not very large.
A-R-S and T-S-D, also instance matching benchmarks, are larger than IIMB, but apart from that they have the same limitations.
STBenchmark can not be fully evaluated using the requirements, because aspects like quality depend on the input ontology (since STBenchmark is a sort of ontology modificator). The biggest disadvantage is, that no reference alignment is generated.

6.5 Summary

The benchmark, presented in this chapter, is constructed to test matching systems, especially with respect to their instance-based matchers. Compared to existing benchmarks, the number of instances is high, but not all concepts contain instances. Hence, the benchmark should be extended in future work. The modifications on the schema level can be improved, i.e. more spelling mistakes can be included or different notations can be used. The structural transformation can also be extended by flattening single entities or by changing data type into object properties and vice-versa. Nevertheless, ONTOBI is more appropriate for testing instance-based matchers than the other existing benchmarks.
In the next chapter, we will use ONTOBI and other test sets to evaluate the novel instance-based matchers on their own and in combination with schema- and structure-based methods in `MICU`.

7 Evaluation and Discussion

In the two Chapters 4 and 5 different approaches to determine entity correspondences have been presented. First of all, two instance-based matchers have been proposed; the first one uses regular expressions or catchwords to find concept and property mappings, the second one calculates features for each property and derives property and concept correspondences. Both approaches are evaluated using the ONTOBI benchmark and additionally using one version of the Islab Instance Matching Benchmark.
With `MICU` we also presented a complete matching system which combines many different matchers and interacts with the user. `MICU` is also evaluated using ONTOBI and additionally, the benchmark tests of the OAEI 2009 are executed.
The remainder of this chapter is organized as follows: The two instance-based matchers are evaluated in Section 7.1. The test data is shortly introduced in Subsection 7.1.1, before presenting the evaluation results of the regular expression approach in Subsection 7.1.2 and of the feature approach in Subsection 7.1.3. Additionally, the results are discussed and some directions for future work are pointed out. Finally, in Section 7.2 `MICU` is evaluated and discussed.

7.1 Instance-based Matchers

7.1.1 Test Data

For testing matching methods we need appropriate test data, which has to fulfill some criteria as described in Section 2.7. The benchmark presented in this thesis, ONTOBI, is designed for this purpose, such that it will be used for evaluating the two instance-based matchers and `MICU`.
For the regular expression matcher, one version of the IIMB is used in addition. IIMB is described in Subsection 3.2.3 and the used version can be found on [IIM10]. The ontologies

included in this benchmark always have the same schema (identical meta information), but the instances differ. This is not a problem, since the instance-based matching methods are first evaluated solely and they include non or only a few pieces of schema information. The reference ontology contains 6 classes, 47 datatype properties and 222 instances. The modifications applied on the reference ontology contain value, structural and logical ones, which provide the basis for different test cases. The tests 02 to 10 contain value transformations, i.e. typographical errors or different data representation formats. Structural transformations like deletion of values or separation of properties are included in the tests 11 to 19. The ontologies of tests 20 to 29 are modified logically, e.g. instances are allocated to different classes. The tests 30 to 37 combine the three transformations. For evaluating our matchers we choose 5 tests of each test group.
The reference alignment only includes instance correspondences but since the schema information is not changed, it is easy to create one.

7.1.2 Regular Expressions Approach

The approach described in Section 4.1.1 uses regular expressions or catchwords to characterize the attributes of a concept using the corresponding instance sets. To show the strengths of this approach and to test different configurations, we evaluate this method using the ONTOBI benchmark. The parameters, that have influence on the matching process and the produced alignment, are:

- regular expression or catchword approach
- number of instances
- similarity measures
- considering of concept instance URIs
- propagation to attributes

First of all, we have to distinguish between the two sub methods, i.e the use of regular expressions or catchwords. The number of instances used for determining the regular expression that best fits to the attributes can influence the matching quality, too. The approach is quite fast, such that bigger amounts of instances do not pose a problem, but the probability of an overfitting might be increased. The number of instances per concept is not too high and it is rather difficult to determine a general best number of instances, such that in most cases we used 200 instances (half of the maximum number of instances per concept).
The similarity measure, that is used to compare the vectors, can be based on the Manhattan or the Euclidean distance, the cosine or another measure.
The regard of labels, i.e. the URIs of concept instances, can be enabled or not. Additionally, the computed concept similarities can be propagated downwards to the attribute level, but since this step is dominated by a string-based label comparison, which is not an instance-

based approach, we only test this step once.

An overview of the different test configurations can be found in Table 7.1. The regular expression list has been created manually for this approach; the catchword list has also been created manually and extended with the 5 most frequent instance values of each attribute of the reference ontology. All tests use the ONTOBI benchmark, because this approach mainly focuses on the detection of concept correspondences, such that the IIMB benchmark with its 7 classes is not suitable.

	RegEx/CW	# inst.	sim. measure	concept inst.	attribute propagation
$R1$	RegEx	200	cosine	no	no
$R2$	RegEx	200	cosine	no	yes
$R3$	RegEx	200	cosine	yes	no
$R4$	RegEx	50	cosine	yes	no
$R5$	RegEx	200	Euclidean	no	no
$R6$	RegEx	200	Euclidean	yes	no
$R7$	RegEx	200	Manhattan	no	no
$R8$	CW	200	cosine	no	no
$R9$	CW	50	cosine	no	no
$R10$	CW	200	Euclidean	no	no

Table 7.1: Overview of the configurations for the regular expression and catchword tests

$R1$ and $R2$ execute the regular expression approach without and with attribute propagation step; the results are shown in Figure 7.1. In general, the precision and recall values are comparable; the most significant difference is observable in tests OS3, OS5, OC2, OC3 and OCC4. In OS5 and OC2 the approach without attribute propagation performs better, which is explainable with the replacement of schema labels by random names. In test OS3 the approach with attribute propagation produces better results, because the instance sets are disjoint but the meta information stays the same.

In the tests $R3$ and $R4$, whose results are presented in Figure 7.2, the number of instances is different, and concepts instances are considered in the matching process. In general, the two configurations produce similar results. The average F-measure of $R4$ is slightly higher with 0.7 compared to 0.68 for $R3$. The tests OS3, OS5, OCC1 and OCC3 provide better results for a lower amount of instances, which may be caused by the disjoint data sets in OS3, OCC1 and OCC3. In OS5 labels have been replaced by random strings, which has influence on the URI of the concept instances. Additionally, the selection of the instances has influence

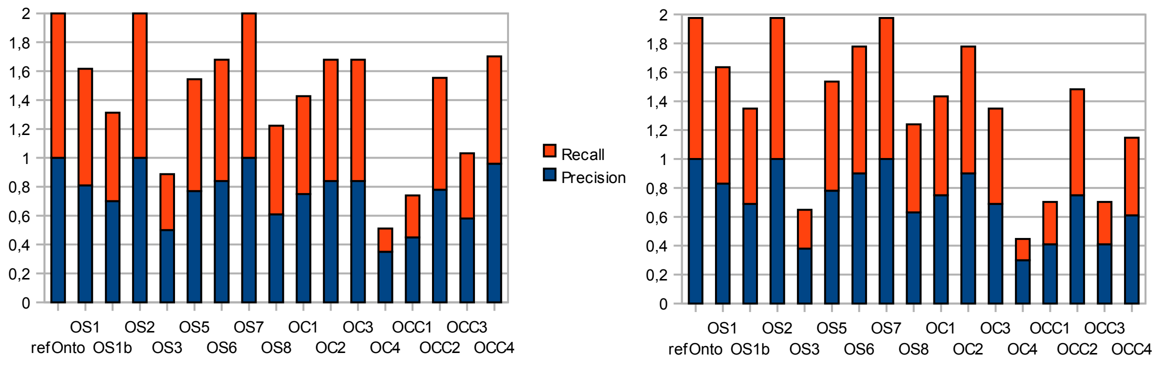


Figure 7.1: left side $R1$, right side $R2$: regular expression approach without and with attribute propagation

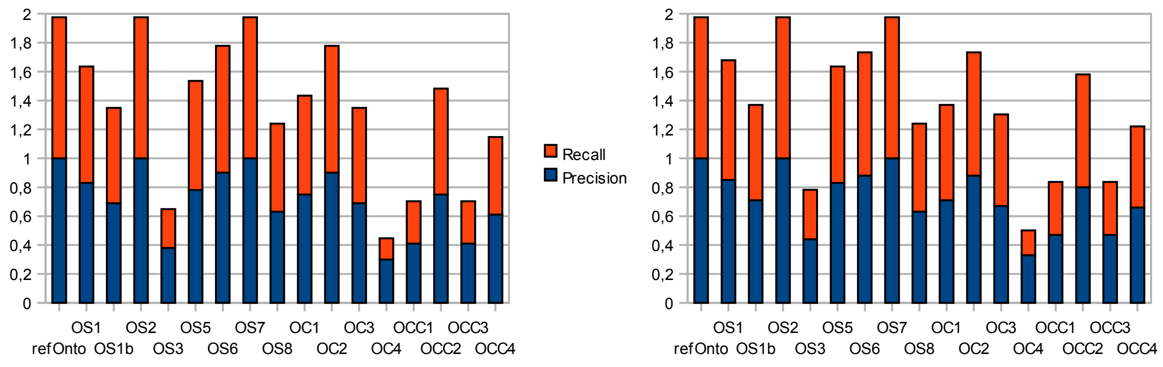


Figure 7.2: left side $R3$, right side $R4$: regular expression approach with 200 or 50 instances and concept instances

on the determination of the assigned regular expression.

In the previous tests, the cosine measure has been used to compare the vectors. In tests $R5$ to $R7$ we use the Euclidean or the Manhattan measure. The distance d calculated by these measures is transformed into a normalized similarity sim_n by using : $sim_n = \frac{1}{1+d}$. We also test the influence on the concept instances if the Euclidean or the Manhattan distance is used. The results are shown in Figure 7.3. In these test cases it makes no difference if the Manhattan or Euclidean measure is used, such that the displayed results are valid for both cases. But there is a difference concerning the use of concept instances. The average F-Measure is 0.62 for R5 and 0.7 for $R6$ but in the test cases OS3, OC2, OC4, OCC1, OCC3 and OCC4 the precision and recall values for the approach that uses the concept instances are significantly lower. Interestingly, this is not the case if the cosine similarity is used (see tests $R1$ and $R3$ for comparison). The addition of a "label" attribute combined with a change of the meta information, which is reflected in the concept instance URI, too, influences the Euclidean similarity significantly.

Finally, we also evaluate the catchword approach. For this purpose three different configurations are used. On the one hand, the size of the used instance set and on the other hand, the similarity measure is varied. The results of the different instance amounts, i.e of

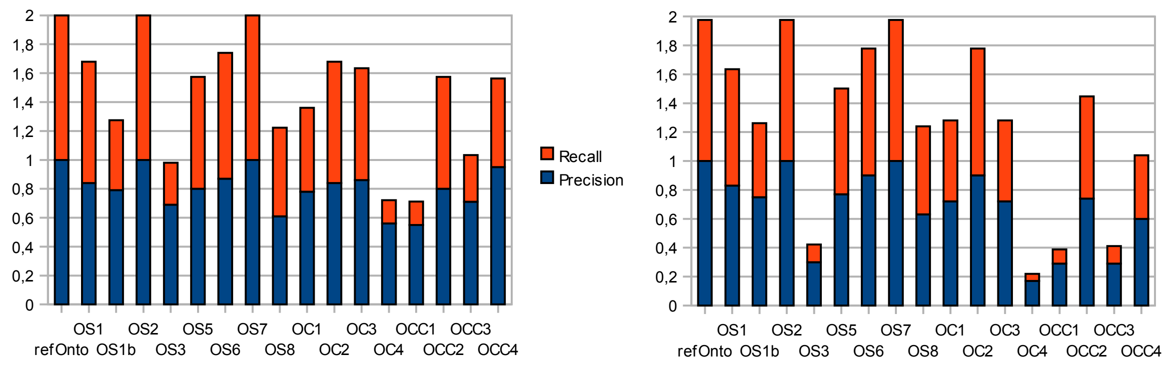


Figure 7.3: left side $R5/R7$, right side $R6$: regular expression approach with Euclidean measure and without or with concept instances

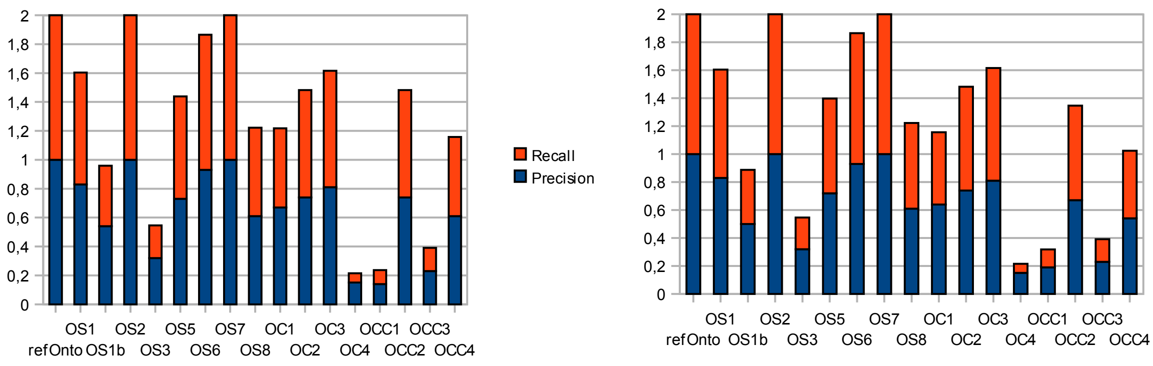


Figure 7.4: left side $R8$, right side $R9$: catchword approach with 200 and 50 instances

tests $R8$ and $R9$, are displayed in Figure 7.4. In contrast to the regular expression approach, the overall result is nearly similar.

As a last test, we execute the catchword approach with the Euclidean measure; the result is presented in Figure 7.5. The overall performance is only slightly lower than with the cosine measure, but in some test cases the results differ a lot. Especially in test $OC1$ (instance misspellings) the result is significantly worse than for the test $R8$.

Discussion

The evaluation shows, that the regular expression approach has a good overall quality. The consideration of concept instances it not always benefiting, especially if the meta information, which is reflected in the URL in most cases, is changed significantly. The cosine measure provides better results than the Euclidean or the Manhattan measure, because it rewards if two vector share any values in the same dimension. The amount of used instances seems not to play a huge role, but to avoid the problem of overfitting, the instance set should not be too large. The catchword approach has been proposed as a more simple approach, such that everybody, who is not an expert in creating regular expressions, can use it. In general, it is worse than the regular expression approach, because especially numeric values can not be

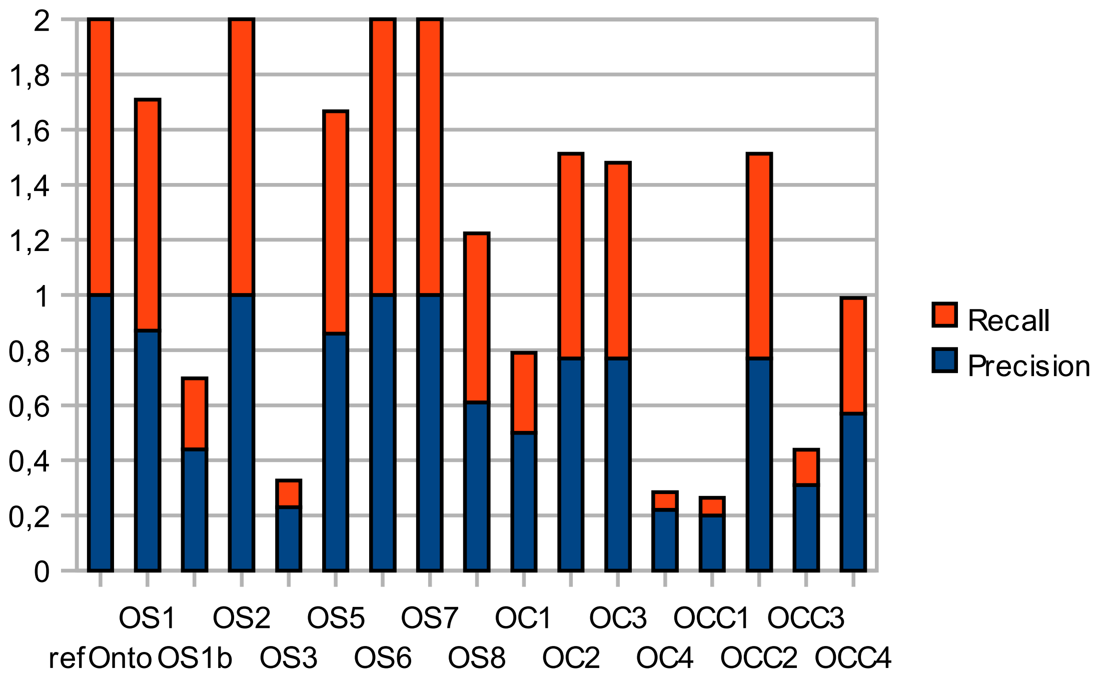


Figure 7.5: $R10$: catchword approach using the Euclidean measure

expressed exactly using catchwords.
In future work, the creation of regular expression and catchword list and the reuse of existent regular expression can be enhanced to provide better results, especially for numeric values. Additionally, another similarity measure could perform better than the cosine measures.

7.1.3 Feature Approach

The instance-based matcher described in Section 4.2 calculates features using the instance sets of attributes. The feature set depends on the type of the instances, which can be string, number or date. Additionally, concept features can be determined and included in the similarity process as well. The following parameters influence the matching process:

- number of instances used for feature calculation
- use of concept features or not
- size of set-based features like frequent values and common substrings
- propagation algorithm
- threshold

In general, it makes sense to use as many instances as possible for calculating the features. But if the instance set is too large, the process may take too much time. At least when computing the common substring values, the number of instances should be limited. Similar

thoughts have to be spent on the size of set-based features.
The concept features provide an additional possibility to increase the similarity of concept pairs that do not provide many instances, but in case of very heterogeneous schema information it might decrease the similarity.
The similarities calculated on the basis of the instance sets are attribute similarities. To find concept correspondences as well the similarity has to be propagated upwards. We proposed two strategies, the maximum and the stable approach.
The threshold determines which similarity value indicates a mapping and might influence the matching result as well.

The feature based approach is evaluated with the IIMB and the ONTOBI benchmarks using different configurations. First, the IIMB tests are described.

Tests with IIMB

As described before, the IIMB ontologies only contain a few classes and properties and the meta information like entity labels is the same for all classes. Hence, the concept-based features would be similar such that we do not consider concept-features in these tests.
The configuration of the test series $T1$ and $T2$ can be found in Table 7.2. The results of both test series, which only differ in the threshold value, are exactly the same. Hence, the result, that can be found in Figure 7.6, is valid for $T1$ and $T2$.

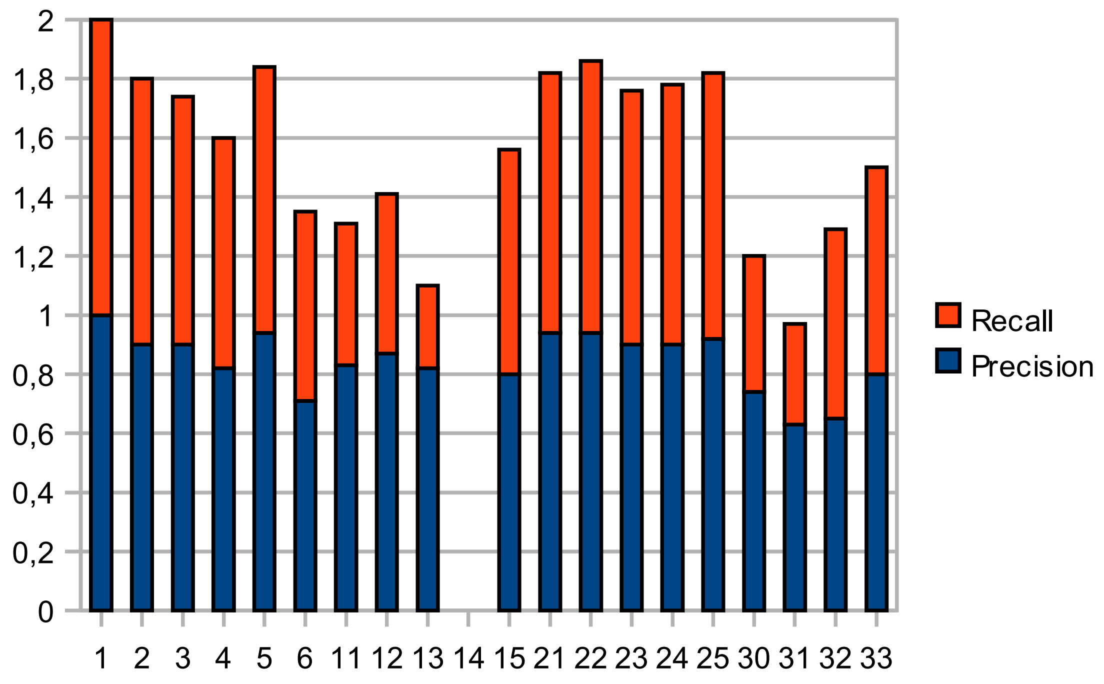


Figure 7.6: $T1/T2$ IIMB: without concept features, threshold 0.5

	# inst.	concept features	size of sets	threshold
$T1$ (IIMB)	all	no	3	0.5
$T2$ (IIMB)	all	no	3	0.7
$T3$ (ONTOBI)	all	no	3	0.3
$T4$ (ONTOBI)	all	no	3	0.5
$T5$ (ONTOBI)	all	no	3	0.7
$T6$ (ONTOBI)	all	yes	3	0.3
$T7$ (ONTOBI)	all	yes	3	0.5
$T8$ (ONTOBI)	all	yes	3	0.5
$T9$ (ONTOBI)	all	no	7	0.5
$T10$ (ONTOBI)	all	no	1	0.5

Table 7.2: Overview of the configurations of the feature approach tests

In tests 01 to 06, the value transformation tests, the approach performs quite well. The average F-measure is 0.87, if we exclude the last test. The values of test 06 include a lot of spelling mistakes in each word, such that the mapping quality is lower than in the other tests.
The structural transformation tests 11 to 15 are rather difficult for this approach. The precision is still acceptable, but the recall especially for test 13 is quite low. In test 13 many datatype properties have been transformed into object properties, and the resource name (that is also included in the matching process) is a senseless identification number. Hence, the object properties decrease the similarity of the entity pairs. In test 14 the feature-based approach fails completely, because all datatype properties, except one called "hasDataValue" which is not included in the reference ontology, have been replaced by object properties.
The logical transformations applied on the ontologies of tests 20 to 25 do not pose a problem for this matching approach, because only the assignment of the instances to the different classes differs. Some classes do not contain instances at all, such that we can not obtain a recall of 1.
The complex tests 30 to 33 combine different transformations. Our approach especially has problems if the spelling mistakes are very grave, which is especially the case in tests 31 and 32.

Tests with ONTOBI

We also evaluate the feature-based approach using the ONTOBI benchmark. All tests are executed using different configurations, which can be found in Table 7.2. Since ONTOBI also

differs the schema information, the concept-based features take part in the matching process as well, but they are only calculated for those concepts that do provide instances.
The distance measure, that compares sets of strings or numbers, has been adapted, such that the number of equal elements is counted and divided by the maximum number of elements in total. Since this produces a similarity, it is transformed into a distance by subtracting the value from 1; more formally i.e.

$$HFD(f_p, g_p) = 1 - \frac{|\{(f,g)|f \in f_p, g \in g_p, f = g\}|}{max(|f_p|, |g_p|)} \tag{7.1}$$

where f_p, g_p are sets. String are compared using the edit distance and the similarity between numbers is calculated with the $ndiff$ measure as described in Subsection 4.2.8.
In the first three tests $T3$ to $T5$ concept features are not calculated and only the threshold is varying. Interestingly, the results of all three tests are (nearly) the same. $T3$ and $T4$ are completely the same, but in some cases of $T5$ one correspondence less is found. Hence, the result for all three tests is shown in Figure 7.7. The recall values correspond to the number of entity correspondences that can be found, i.e. concepts and properties, that do not contain instances, are not considered.
As expected, the most difficult tests have been the ones in which the instance set is completely different (tests OS3, OCC1 and OCC3). Additionally, the flattening of the hierarchy seems to be hard to cope with (test OS8,OC4,OCC3).
The same tests have been repeated including the concept features ($T6$ to $T8$), the results can be found in Figure 7.8. Equally to the concept test, the threshold has been varied from 0.3 to 0.7 and no significant differences occur. The most interesting observation is, that the quality of the matching result is decreased when including the concept features. The precision is always lower in contrast to the tests $T3$ to $T5$, only the recall value is better in some cases.

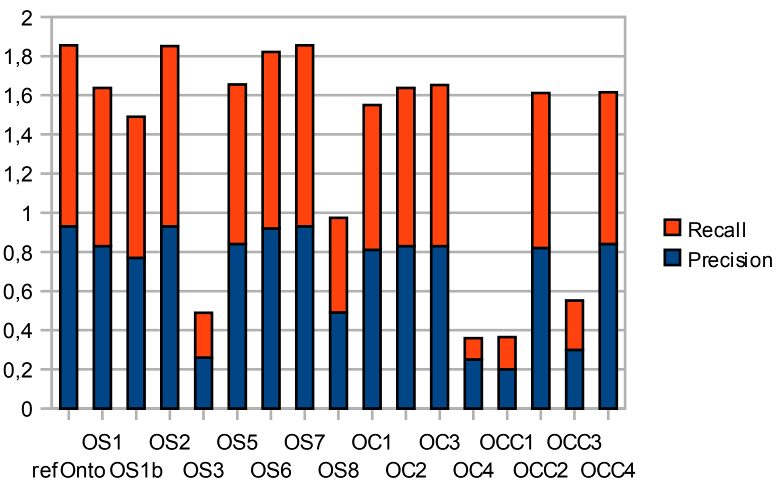


Figure 7.7: $T3$ to $T5$ ONTOBI: without concept features, threshold $0.3, 0.5, 0.7$

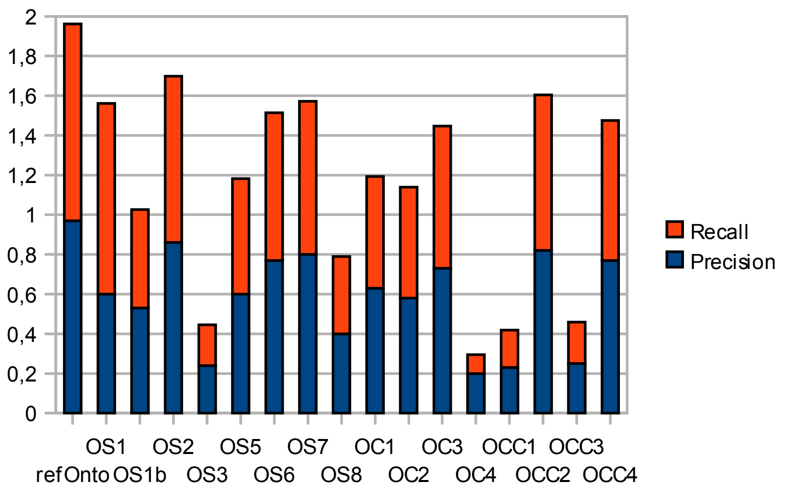


Figure 7.8: $T6$ to $T8$ ONTOBI: with concept features, threshold $0.3, 0.5, 0.7$

The size of the several set-based features like frequent values can also influence the matching process. Hence, we test our approach again with varying size. The tests described before always calculate 3 values, the test $T9$ calculates 7 values, $T10$ only 1 value; concept features are also omitted. The results are displayed in Figure 7.9 and 7.10. The increase of the set size results in a worse result, but the reduction of the size to 1 slightly enhances the matching quality in contrast to the tests with set size 3.

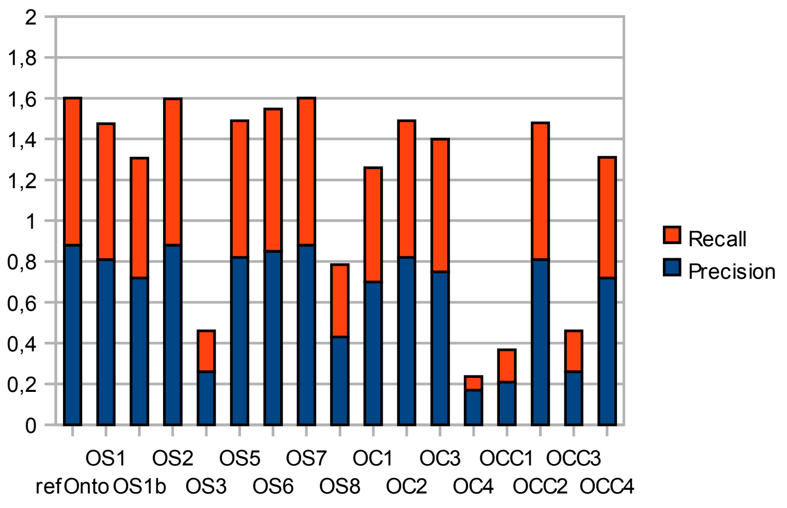


Figure 7.9: $T9$ ONTOBI: without concept features, threshold 0.5, set size 7

7.1.4 Discussion

The results of the evaluation show that the feature-based matching approach is very stable according to the threshold, i.e. the differentiation of similar and dissimilar entities is good. The precision and recall values of the reference ontology are not 1, because some attribute

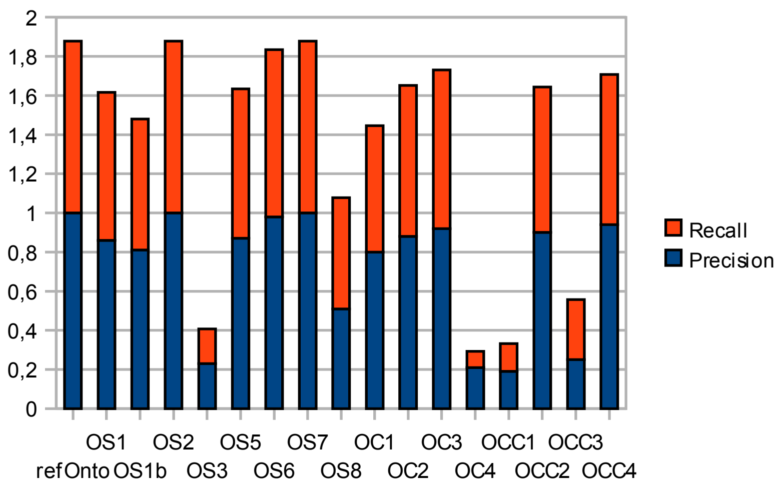


Figure 7.10: $T10$ ONTOBI: without concept features, threshold 0.5, set size 1

pairs share the same feature vector, such that the similarity is the same. In this case, only the correspondence, that is found first, is considered. It works well for overlapping instance sets, which should be the most common case, i.e. it is not very probable that the instance sets are disjoint. The number of elements calculated for the set-based measures has to be adapted to the ontology set. It seems to be more effective, if the set size is rather small. Concept features do not increase the matching quality if the meta information is heterogeneous for most entities. The flattening of the hierarchy seems to be quite problematic for the approach, such that it has to be enhanced in this direction. Overall, the results are promising, especially under the consideration that this approach should be used as an extension to and in combination with schema-based matching approaches.
In future work the features can be used as input e.g. for a neural network that classifies the instance sets or the similarity function can be improved.

7.1.5 Comparison of Regular Expression and Feature Matcher

In general, the two approaches are comparable, i.e. the problems mainly occur in the same test cases. In test OS1b, which includes spelling mistakes in the instances, the regular expression approach performs slightly worse, because the spelling mistakes influence the assignment to more specialized regular expressions.
The average F-measure of the regular expression approach is higher than for the feature matcher, which is caused by the exactness of the regular expressions in contrast to the feature values, which might be similar for different attributes (e.g. the average length). Additionally, the regular expression approach is faster.

The advantage of the feature matcher is that it does not need any additional effort before the execution, in contrast to the regular expression approach, for which first regular expressions have to be defined. This disadvantage of the regular expression approach might become

smaller by time, because regular expressions (or lists) can be reused.
Both approaches need propagation strategies. The feature approach calculates property mappings and propagates the corresponding similarities to the concept level, whereas the regular expression approach propagates the concept similarities to the properties.

It is difficult to define exact regular expressions for numeric values. In the case that the instances of an ontology contain many numbers, the feature matcher should produce better results. The same holds for instances that include many dates, because they cannot easily be distinguished by regular expressions. Contrarily, for string attributes, the regular expression approach can capture the semantics more precisely.

Concluding we can state that both approaches have advantages and disadvantages and that the performance of both methods depends on the matching task.

7.2 MICU

`MICU` is a complete matching system, which combines the instance-based matcher described in this thesis with schema- and structure-based matchers. `MICU` is also tested using ONTOBI and, additionally, some of the OAEI 2009 benchmark tests are executed.
The general flow of `MICU` is described in Section 5.1; first schema-based matchers are applied on the schema. Afterwards, the instance-based matcher (in general the feature matcher) finds additional property matches, and finally, the similarity is propagated to unmatched entities. The general idea is to compare the whole schema using the schema-based matchers. But if the ontologies of ONTOBI are used, the schema-based matching of all entities would last too long. For this purpose we adapted our system such that the result is produced within a shorter time period. The OAEI tests pose another challenge, because in these cases the ontologies are quite small, and there are only a few instances available.
In general, the results of the matching processes can be influenced by some factors:

- used matchers
- threshold and weights
- time optimization
- user feedback

The used matchers can vary according to the input ontology. Additionally, thresholds and weights have to be set for several parts of the program. Time optimization is an issue, that appeared when we tested `MICU` with ONTOBI (see next subsection for details). In the following, the different test scenarios and the adaption we have to make on `MICU` are explained.

7.2.1 Tests with ONTOBI

The ontologies of ONTOBI are very large; the reference ontology contains 213 classes, 1181 object properties and 1103 data types properties. In the original architecture of `MICU` all entities are compared to at least all entities of similar type using the schema-based matchers. In this case it would result in $213 \cdot 213 + 1.181 \cdot 1.181 + 1.103 \cdot 1.103 = 2.656.739$ comparisons to be made, which would cost a lot of time. Especially when involving synonym and comment sources (WordNet or databases) the execution time is too long. Hence, we adapted `MICU` as follows: first, all concepts are matched using the schema-based matching methods, because the concepts form the biggest information unit. Afterwards, the instance-based matcher finds object and datatype property mappings. The concept propagation step of the feature matcher can determine additional concept correspondences. The structure-based matcher propagates the concept similarities to unmatched concept pairs. Finally, there is another schema-based matching round in which the properties of matched concepts are compared and aligned.
We test `MICU` using the configurations described in Table 7.3. A detailed description of the different configurations will be given in the following together with the presentation of the results.

	used matchers	thresholds	time opt.	user
$M1$	feature matcher	0.7	yes	no
$M2$	feature matcher + concept propagation	0.7	yes	no
$M3$	feature matcher	0.7	yes	yes
$M4$	regular expression	0.7	yes	no
$M5$	no instance matcher	0.5	yes	no

Table 7.3: Overview of the configurations of the ONTOBI/MICU test

In general, the threshold of the schema-based matcher and the feature matcher is set to 0.7 for most of the tests. As shown before, the threshold does not play a significant role for the feature matcher, since this approach has a good discriminatory power, i.e. similar entities have a high similarity. The schema-based matchers also showed a good accuracy in the ONTOBI tests, such that the threshold is set to 0.7 as well. A lowering of the threshold would produce very similar results, because for each entity only the best fitting correspondence is searched, which has a similarity of 0.7 or higher. The structure-based matcher has a lower threshold in all tests, because we aim to find the best partner for all unmatched concepts, such that preferably there is a mapping partner for each entity.
Tests $M1$ and $M2$ combine schema- and structure-based matcher with the feature matcher. In the first case, the concept propagation step of the feature matcher is disabled, i.e. the feature matcher can only find attribute correspondences. This is reflected in the results as shown in Figure 7.11. The results of $M2$ are around 0.2 higher than those of $M1$ in each test

case, besides OS5 and OC2 (the random names cases). The concept propagation algorithm finds a few more correspondences for OS5 and OC2, but the number of concepts, that include instances, is too small in contrast to the number of total entities, such that the precision and recall values do not differ much.

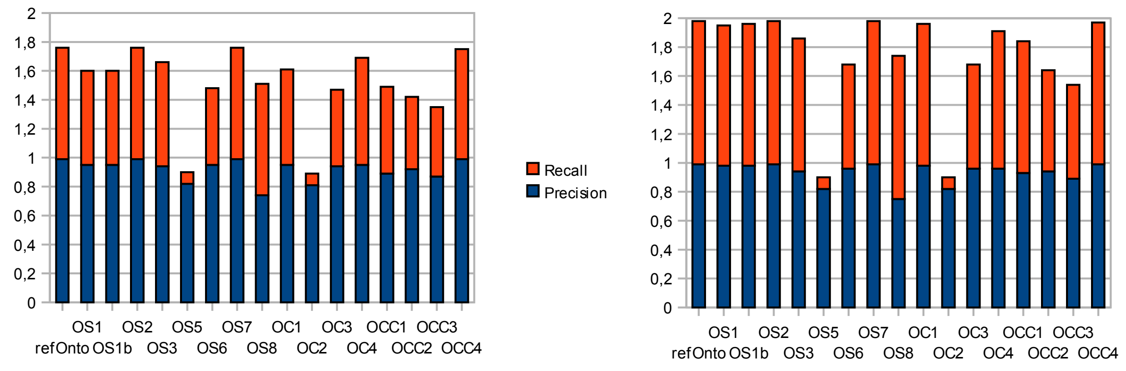


Figure 7.11: M1 and M2, ONTOBI: feature matcher without and with concept propagation

In test $M3$ the user feedback dialog is evaluated. The threshold is again 0.7 and the user is asked to look at correspondences that have a similarity between 0.3 and 0.7. The result is shown in Figure 7.12. The results does not differ significantly compared to $M2$, because, as explained before, the discriminatory power of the used algorithms is very high. In case of spelling mistakes (tests OS1, OS1b) and synonyms (tests OS6, OC3, OCC3) some more correspondences can be found. Whenever entity names are replaced by random strings, the user gets many correspondence proposals, but it is difficult to decide which correspondences are correct. In all other cases, the inclusion of the user is very low, i.e. he does not have to look at many correspondences (around 20 in the maximum, in all tests besides the random tests).

We also test the quality of `MICU` in the case, that the feature matcher is replaced by the regular expression matcher. In test $M4$ we used the same regular expression file as in the evaluation of the regular expression matcher described in Subsection 7.1.2. The user dialog is enabled, and the attribute propagation step of the regular expression matcher is enabled, i.e. after determining concept mappings, the attributes are compared using their corresponding regular expression and their string similarity. Figure 7.13 shows the performance of `MICU` in this test. For most of the tests cases, the regular expression approach performs slightly better than the feature approach. Especially precision and recall of tests OS3, OCC1 and OCC3 are higher. In contrast, the results of the random name tests OS5 and OC2 are lower; this is caused by the attribute propagation step of the regular expression. The string similarity comparison is not useful in this case, such that the comparison of the regular expressions dominates the similarity of two attributes. Since the number of regular expression included in the list is limited, it is not unlikely that two attributes share the same regular expression although they are not similar.

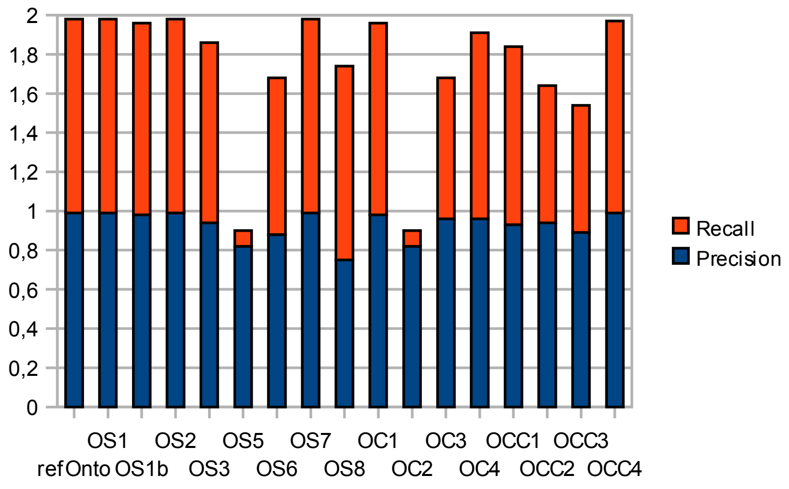


Figure 7.12: M3 ONTOBI: feature matcher with user feedback

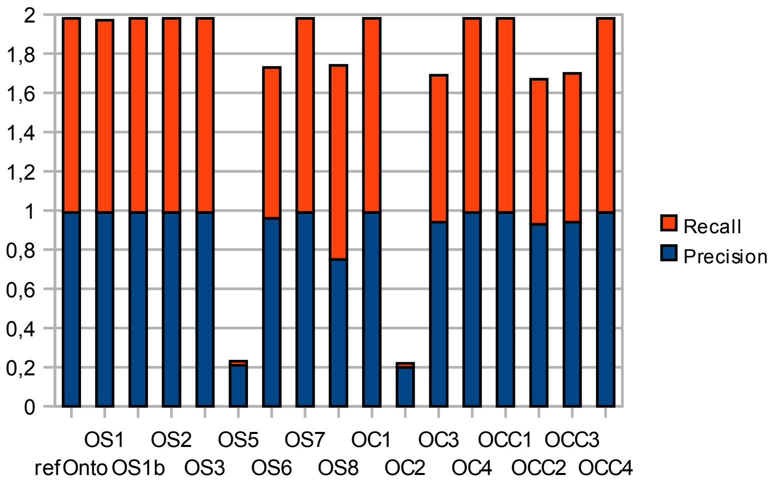


Figure 7.13: M4 ONTOBI: regular expression matcher

Finally, the instance-based matchers of `MICU` are disabled completely in test $M5$. The result of this test is shown in Figure 7.14. The precision and recall values of most test cases are slightly smaller compared to test $M2$. The results of OC4 and OCC3 are significantly better, because the names are not changed in test OC4, and in OCC3 the instance sets differ completely, which reduces the performance of the feature matcher. The result for OCC3 with the regular expression matchers in $M4$ is comparable to the values reached in $M5$. In the random names test, the schema-based matchers fail completely.

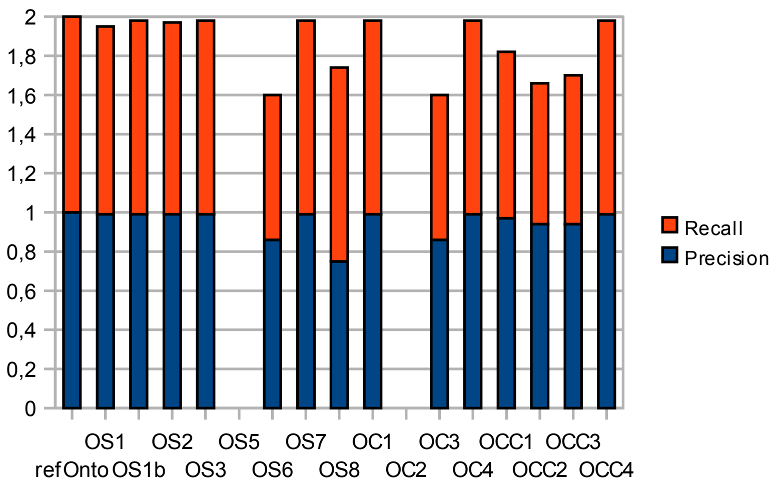


Figure 7.14: M5 ONTOBI: no instance matcher

7.2.2 Tests with OAEI

Besides testing MICU and especially the combination of the different matchers, we wanted to compare MICU to other matching systems. One good possibility is to perform the OAEI tests, see Subsection 3.2.1. We decided to choose some of the benchmark test 2009, where we concentrate on the most interesting ones. The performed tests are shortly described in Table 7.4.

The tests, that are not executed in this evaluation are mainly combinations of the other tests, such that their executions would not provide significantly new information. We also used different configurations of MICU for performing the OAEI tasks; an overview can be found in Table 7.5.

First of all, we compared the performance of MICU with and without instance matcher. The results are displayed in Figure 7.15.

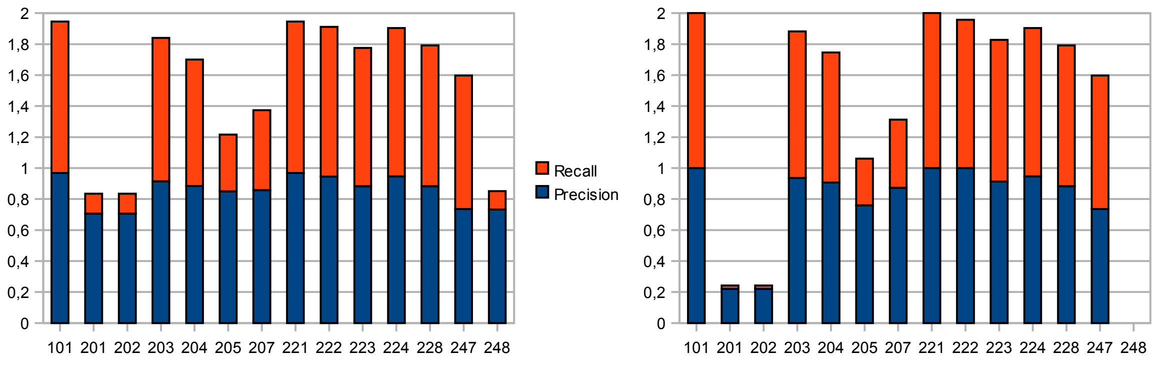


Figure 7.15: $M6$ and $M7$ OAEI: with feature matcher and without any instance-based matcher

test number	description
101	reference ontology
201	random names
202	random names, no comments
203	no comments
204	naming convention
205	synonyms
207	translation to French
221	no hierarchy
222	flattened hierarchy
223	expanded hierarchy
224	no instances
228	no properties
247	expanded hierarchy, no instances, no properties
248	random names, no comments, no hierarchy

Table 7.4: OAEI benchmark tests

As expected, in test cases in which the meta information is changed significantly (tests 201, 202 and 248) the instance-based matcher can find additional correspondences and hence increase recall and precision. The same observation can be made for the synonym test 205 and the French ontology in test 207. In test 248 the instance-matcher is essential to find any matching, because schema and structure information is changed or suppressed completely. In all other cases, `MICU` gets similar or slightly better results without the instance matchers. This is caused by the fact, that there are only a few instances that are quite similar to each other, such that it is difficult to distinguish between the sets. The average precision of $M6$ (without test 101) is 0.85, the corresponding recall value is 0.66. In comparison, the values of test configuration $M7$ (without test 101) are 0.72 for precision, and 0.63 for recall. Hence, the use of instance matchers is helpful, even though the set of instances is quite small.
We also tested `MICU` with the regular expression matcher instead of the feature matcher in test $M8$. The result is presented in Figure 7.16. Interestingly, the precision value is higher than for test $M6$; the average value of the $2xx$ tests is 0.9. The average recall value is only slightly smaller than that of $M6$ and is 0.64. In this test configuration of `MICU`, the regular expression matcher is used with enabled attribute propagation, i.e. for matched concepts the attributes are matched using their label and the assigned regular expression.

	used matchers	thresholds	time opt.	user feedback
$M6$	feature matcher	0.7	no	no
$M7$	no instance-based matcher	0.7	no	no
$M8$	regular expression	0.7	no	no
$M9$	feature matcher	0.7	no	yes

Table 7.5: Overview of the configurations of OAEI/MICU test

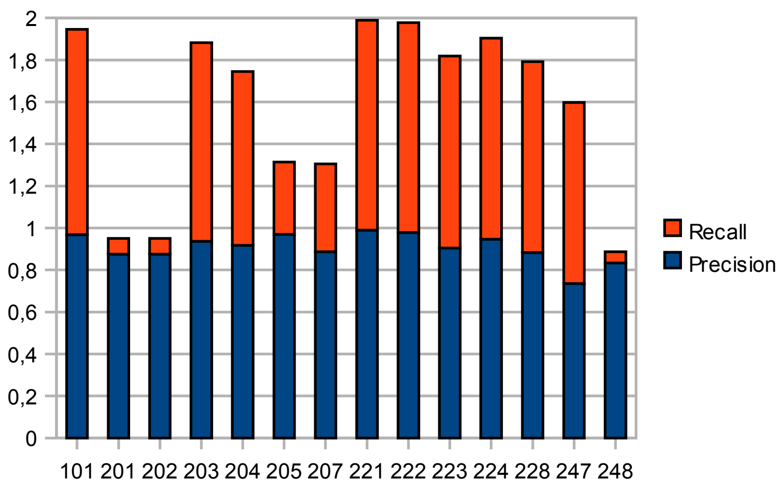


Figure 7.16: $M8$ OAEI: regular expression matcher

The precision of the “difficult” tests 201, 202 and 248 is higher, but the recall is lower. The precision in the synonym test 205 is significantly higher, which might be caused by the attribute propagation step of the instance-based matcher.

Finally, we evaluated the OAEI tests with enabled user interaction; the result of test $M9$ is shown in Figure 7.17.

7.2.3 Discussion

The performance of `MICU` in the ONTOBI tests is quite good. Especially the precision is consistently high, the recall value can be improved. The results in test $M5$ shows, that the modifications on the schema level of ONTOBI do not pose a huge problem for the schema matchers of `MICU`. Hence, ONTOBI could be improved such that it is a bigger challenge. For comparison purposes, ONTOBI should have been tested with other systems as well. Unfortunately, none of the available matching systems was able to cope with the huge amount of data.

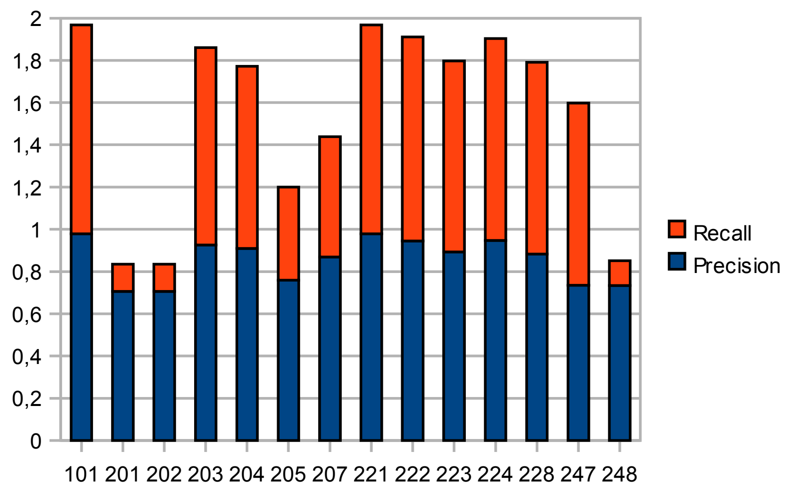


Figure 7.17: $M9$ OAEI: user feedback

In the case that entity names are senseless or very different, a user feedback dialog could improve the results. At present, `MICU` only displays entity names in the user feedback dialog, which makes it very difficult for the user to decide which correspondence might be the right one. In future work, the user feedback dialog can be enriched with further information, e.g. instance samples, comments, or the position of the entity within the ontology.

The results of the OAEI 2009, from which the OAEI tests executed with `MICU` have been taken, can be found in [EFH+09]. For the $2xx$ tests, the precision values are in the range of $[0.73, 0.98]$ and the recall values range from 0.23 to 0.86. Hence we can state, that our system can produce appropriate results (although we have not executed all tests), even though `MICU` can not compete with the best systems. Due to the facts, that `MICU` needs instances to gain best results in case of dissimilar schema information and that these are not sufficiently available in the OAEI tests, the result is satisfying.

8 Conclusion and Future Work

This chapter concludes this thesis. First, a summary with the contributions of this thesis is given in Section 8.1. Finally, some ideas for future work are presented in Section 8.2.

8.1 Summary

Heterogeneous ontologies can be found in many different areas. For several applications it is necessary to match those ontologies, preferably automatically. Existing matching systems mostly rely on schema and structure information, but the instances should also be considered, because the information content of the instance set is not negligible.
In this thesis, we presented two novel instance-based matchers, which exploit the instance sets in two different ways. The first matcher uses regular expressions and catchwords to describe attributes and hence concepts. The concept representations are used to determine concept correspondences, which provide the basis for an additional attribute matching step. The disadvantage of this approach is, that prior effort is needed to create the regular expressions. The second approach does not need any input of the user. The instance set of each attribute is used to calculate a set of features, which is different for numeric, string or date values. Additionally, concept features representing some pieces of schema information can be calculated. The features of attributes having the same type are then compared using a heterogeneous similarity function and finally attribute correspondences are determined. The attribute similarities can then be propagated to the concept level.
Although instance-based methods provide good results, they should be combined with common schema- and structure-based matchers. Instance-based methods are quite time-consuming, and in general not all concepts of an ontology provide instances. Hence, we presented `MICU`, a matching system that combines schema-, instance- and structure-based methods to determine a mapping between different kinds of ontologies (with or without instances etc.). Additionally, alignments can be reused and WordNet can be queried to obtain comments or synonyms.

MICU is extended by an efficient user feedback dialog.
For testing single matching methods or complete matching systems an appropriate benchmark is needed. The existing benchmarks do not fulfill the demands we make on test sets. Hence, we developed an additional benchmark named ONTOBI, which contains much more entities and instances than all other benchmarks. Using this benchmark, we evaluated our two instance-based matchers and MICU.
The evaluation showed, that the instance-based matchers are comparable and both have a good performance. The same observation can be made for MICU.

8.2 Future Work

The field of matching heterogeneous information sources has been studied extensively in the last decades. Ontology matching is a more recent issue, and there is still some work to do.
The instance-based matchers presented in this thesis can be enhances, especially with respect to the used similarity functions. Additionally, the features could be used to train neural networks or other models used for classification.
The combination of different kinds of matchers is also an issue for which more work is needed. An idea is to automatically determine, which matcher is applied on which entity set of the ontology, such that the best result is achieved.
Another remarkable point for future work is the enhancement of existing evaluation benchmarks. The ontologies of ONTOBI provide a good basis, but the modifications have to be extended.
Furthermore, there is a need to adapt existing matching systems, such that they can process huge ontologies as the ones included in ONTOBI.

References

[ATV08] Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. STBenchmark: Towards a benchmark for mapping systems. *Proc. VLDB Endow.*, 1(1):230–244, 2008.

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised) . `http://www.w3.org/TR/rdf-syntax-grammar/`, February 2004.

[BG04] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. `http://www.w3.org/TR/rdf-schema/`, February 2004.

[BGNV08] Geert Jan Bex, Wouter Gelade, Frank Neven, and Stijn Vansummeren. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 825–834, New York, NY, USA, 2008. ACM.

[BL] Tim Berners-Lee. What the Semantic Web can represent. http://www.w3.org/DesignIssues/RDFnot.html,.

[BLG05] Tim Berners-Lee and Networking Group. Uniform Resource Identifier (URI): Generic Syntax . `http://www.ietf.org/rfc/rfc3986.txt`, January 2005.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[BM02] Jacob Berlin and Amihai Motro. Database Schema Matching Using Machine Learning with Feature Selection. In *Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002, Toronto, Canada, May 27-31, 2002, Proceedings*, pages 452–466, 2002.

[BMKL02] Denilson Barbosa, Alberto O. Mendelzon, John Keenleyside, and Kelly Lyons. ToXgene: An extensible template-based data generator for XML. In *In WebDB*, pages 49–54, 2002.

[BN05] Alexander Bilke and Felix Naumann. Schema Matching using Duplicates. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 69–80, 2005.

[BSZ03] Paolo Bouquet, Luciano Serafini, and Stefano Zanobini. Semantic Coordination: A New Approach and an Application. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, pages 130–145, 2003.

[CFM06] Silvana Castano, Alfio Ferrara, and Stefano Montanelli. Matching Ontologies in Open Networked Systems: Techniques and Applications. pages 25–63, 2006.

[CH98] William Cohen and Haym Hirsh. Joins that Generalize: Text Classification Using WHIRL. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 169–173, 1998.

[DDL00] AnHai Doan, Pedro Domingos, and Alon Y. Levy. Learning Source Description for Data Integration. In *WebDB (Informal Proceedings)*, 2000.

[DGB07] Jérôme David, Fabrice Guillet, and Henri Briand. Association Rule Ontology Matching Approach. *International Journal on Semantic Web and Information Systems*, 3(2):27–49, 2007.

[DLD$^+$04] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Y. Halevy, and Pedro Domingos. iMAP: Discovering Complex Mappings between Database Schemas. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 383–394, 2004.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between Ontologies on the Semantic Web. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 662–673. ACM Press, 2002.

[DMDH04] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y. Halevy. Ontology Matching: A Machine Learning Approach. In *Handbook on Ontologies*, pages 385–404. Springer, 2004.

[DMQ03] Dejing Dou, Drew Mcdermott, and Peishen Qi. Ontology Translation on the Semantic Web. In *Journal of Data Semantics*, page 2005, 2003.

[DR02] Hong Hai Do and Erhard Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 610–621, 2002.

[EE05] Marc Ehrig and Jérôme Euzenat. Relaxed Precision and Recall for Ontology Matching. In *Integrating Ontologies '05, Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies, Banff, Canada, October 2, 2005*, 2005.

[EFH$^+$09] Jérôme Euzenat, Alfio Ferrara, Laura Hollink, Antoine Isaac, Cliff Joslyn, Véronique Malaisé, Christian Meilicke, Andriy Nikolov, Juan Pane, Marta Sabou, François Scharffe, Pavel Shvaiko, Vassilis Spiliopoulos, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, Vojtech Svátek, Cássia Trojahn dos Santos, George A. Vouros, and Shenghui Wang. Results of the Ontology Alignment Evaluation Initiative 2009. In *Proceedings of the 4th International Workshop on Ontology Matching (OM-2009) collocated with the 8th International Semantic Web Conference (ISWC-2009) Chantilly, USA, October 25, 2009*, 2009.

[EM07] Daniel Engmann and Sabine Maßmann. Instance Matching with COMA++. In *Datenbanksysteme in Business, Technologie und Web (BTW 2007), Workshop Proceedings, 5.-6. März 2007, Aachen, Germany*, 2007.

[EP06] Andy Seaborne Eric Prud'hommeaux. SPARQL Query Language for RDF . `http://www.w3.org/TR/rdf-sparql-query/`, October 2006.

[ES04] Marc Ehrig and Steffen Staab. QOM - Quick Ontology Mapping. In *INFORMATIK 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Ulm, 20.-24. September 2004*, pages 356–361. GI, 2004.

[ES07] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching.* Springer-Verlag, Heidelberg (DE), 2007.

[Euz94] Jérôme Euzenat. Brief overview of T-tree: the Tropes Taxonomy building Tool. In *Proceedings of the 4th ASIS SIG/CR workshop on classification research , Columbus (OH US)*, pages 69–87, 1994.

[Euz06] Jérôme Euzenat. An API for ontology Alignment (version 2.1). http://gforge.inria.fr/docman/view.php/117/251/align.pdf, 2006.

[Euz07] Jérôme Euzenat. Semantic Precision and Recall for Ontology Alignment Evaluation. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 348–353, 2007.

[EV04] Jérôme Euzenat and Petko Valtchev. Similarity-Based Ontology Alignment in OWL-Lite. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 333–337, 2004.

[Fle73] J.L. Fleiss. *Statistical Methods for Rates and Proportions.* John Wiley and Sons, 1973.

[FLMV08] Alfio Ferrara, Davide Lorusso, Stefano Montanelli, and Gaia Varese. Towards a Benchmark for Instance Matching. In *Proceedings of the 3rd International*

Workshop on Ontology Matching (OM-2008) Collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26, 2008, 2008.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GS62] D. Gale and L. S. Shapley. College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[Hab04] Mehran Habibi. *Real World Regular Expressions with Java 1.4.* APress, 2004.

[Hea09] Marti A. Hearst. *Search User Interfaces.* Cambridge University Press, 2009.

[Hum83] Robert A. Hummel. A Design Method for Relaxation Labeling Applications. In *Proceedings of the National Conference on Artificial Intelligence. Washington, D.C., August 22-26, 1983*, pages 168–171, 1983.

[IIM10] The ISLab Instance Matching Benchmark. http://islab.dico.unimi.it/iimb/, last visited: October 2010.

[ITH03] Ryutaro Ichise, Hideaki Takeda, and Shinichi Honiden. Integrating Multiple Internet Directories by Instance-based Learning. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 22–30, 2003.

[JHCQ05] Ningsheng Jian, Wei Hu, Gong Cheng, and Yuzhong Qu. Falcon-AO: Aligning Ontologies with Falcon. In *Integrating Ontologies '05, Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies, Banff, Canada, October 2, 2005*, 2005.

[Jie04] Jie Tang and Bangyong Liang and Juan-Zi Li and Kehong Wang. Risk Minimization Based Ontology Mapping. In *Content Computing, Advanced Workshop on Content Computing, AWCC 2004, ZhenJiang, JiangSu, China, November 15-17, 2004, Proceedings*, pages 469–480, 2004.

[JMSK09] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology matching with semantic verification. *Journal of Web Semantics*, 7(3):235–251, 2009.

[Kin80] Ross Kindermann. *Markov Random Fields and Their Applications (Contemporary Mathematics ; V. 1).* American Mathematical Society, 1980.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4), 1995.

[KQL07] David Kensche, Christoph Quix, Xiang Li 0002, and Yong Li. GeRoMeSuite: A System for Holistic Generic Model Management. In *Proceedings of the 33rd*

International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007, pages 1322–1325, 2007.

[LBK$^+$09] Jens Lehmann, Chris Bizer, Georgi Kobilarov, Søren Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A Crystallization Point for the Web of Data. *Journal of Web Semantics*, 2009.

[LC94] Wen-Syan Li and Chris Clifton. Semantic Integration in Heterogeneous Databases Using Neural Networks. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 1–12, 1994.

[LC00] Wen-Syan Li and Chris Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks, journal = Data and Knowledge Engineering. 33(1):49–84, 2000.

[LG01] Martin S. Lacher and Georg Groh. Facilitating the Exchange of Explicit Knowledge through Ontology Mappings. In *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference, May 21-23, 2001, Key West, Florida, USA*, pages 305–309, 2001.

[LT06] Patrick Lambrix and He Tan. SAMBO - A system for aligning and merging biomedical ontologies. *Journal of Web Semantics*, 4(3):196–206, 2006.

[MBR01] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic Schema Matching with Cupid. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 49–58, 2001.

[MER06] Sabine Massmann, Daniel Engmann, and Erhard Rahm. COMA++: Results for the Ontology Alignment Contest OAEI 2006. In *Proceedings of the 1st International Workshop on Ontology Matching (OM-2006) Collocated with the 5th International Semantic Web Conference (ISWC-2006), Athens, Georgia, USA, November 5, 2006*, 2006.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*, pages 117–128. IEEE Computer Society, 2002.

[MYC08] Erwan Moreau, François Yvon, and Olivier Cappé. Robust Similarity Measures for Named Entities Matching. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 593–600, Manchester, UK, August 2008.

[n306] Notation3. http://www.w3.org/DesignIssues/Notation3, last update: March 2006.

[NVVM06] Miklos Nagy, Maria Vargas-Vera, and Enrico Motta. DSSim-ontology Mapping with Uncertainty. In *Proceedings of the 1st International Workshop on Ontology Matching (OM-2006) Collocated with the 5th International Semantic Web Conference (ISWC-2006), Athens, Georgia, USA, November 5, 2006*, 2006.

[OAE09] Ontology Alignment Evaluation Initiative - OAEI-2009 Campaign. http://oaei.ontologymatching.org/2009/, 2009.

[Pro09] The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/, December 2009.

[QGK09] Christoph Quix, Sandra Geisler, David Kensche, and Xiang Li 0002. Results of GeRoMeSuite for OAEI 2009. In *Proceedings of the 4th International Workshop on Ontology Matching (OM-2009) collocated with the 8th International Semantic Web Conference (ISWC-2009) Chantilly, USA, October 25, 2009*, 2009.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[SA09] Md. Hanif Seddiqui and Masaki Aono. An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):344 – 356, 2009.

[SCW04] Michael K. Smith and and Deborah L. McGuinness Chris Welty. OWL Web Ontology Language Guide. http://www.w3.org/TR/owl-guide/, last update: February 2004.

[SE05] Pavel Shvaiko and Jérôme Euzenat. A Survey of Schema-Based Matching Approaches. In *Journal on Data Semantics IV*, volume 3730 of *Lecture Notes in Computer Science*, pages 146–171. Springer, 2005.

[SFW83] Gerard Salton, Edward A. Fox, and Harry Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26:1022–1036, November 1983.

[SM01] Gerd Stumme and Alexander Maedche. FCA-MERGE: Bottom-Up Merging of Ontologies. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 225–234, 2001.

[WES08] Shenghui Wang, Gwenn Englebienne, and Stefan Schlobach. Learning Concept Mappings from Instance Similarity. In *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, pages 339–355, 2008.

[WM97] D. Randall Wilson and Tony R. Martinez. Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, cs.AI/9701101, 1997.

[wol10] About Wolfram—Alpha. `http://www.wolframalpha.com/about.html`, last visited: October 2010.

[Wor10] WordNet . http://wordnet.princeton.edu/, last visited: October 2010.

[WX09] Peng Wang and Baowen Xu. Lily: Ontology Alignment Results for OAEI 2009. In *Proceedings of the 4th International Workshop on Ontology Matching (OM-2009) collocated with the 8th International Semantic Web Conference (ISWC-2009) Chantilly, USA, October 25, 2009*, 2009.

[Zai08a] Katrin Zaiß. Entwicklung eines Frameworks für instanzbasiertes Ontologie-Matching. In *Tagungsband zum 20. GI-Workshop über Grundlagen von Datenbanken (20th GI-Workshop on the Foundations of Databases), Apolda, Thüringen, 13.-16. Mai 2008*, 2008.

[Zai08b] Katrin Zaiß. Ontologie-Matching: Überblick und Evaluation. *Datenbank-Spektrum*, 8(24):12–17, 2008.

[ZC09a] Katrin Zaiß and Stefan Conrad. Instance-Based Ontology Matching Using Different Kinds Of Formalisms. In *Proceedings of World Academy of Science, Engineering and Technology, International Conference on Semantic Web Engineering, July 29-31, Oslo, Norway*, pages 164–172, 2009.

[ZC09b] Katrin Zaiß and Stefan Conrad. Partial Ontology Matching Using Instance Features. In *On the Move to Meaningful Internet Systems: OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part II*, pages 1201–1208. Springer, 2009.

[ZCV10] Katrin Zaiß, Stefan Conrad, and Sven Vater. A Benchmark for Testing Instance-Based Ontology Matching Methods. In *Proceedings of 17th International Conference on Knowledge Engineering and Knowledge Management, 11th October-15th October 2010, Lisbon, Portugal*, 2010.

[ZSC08] Katrin Zaiß, Tim Schlüter, and Stefan Conrad. Instance-Based Ontology Matching using Regular Expressions. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2008 Workshops, ODBase 2008, LNCS 5333, 9-14. November 2008, Monterrey, Mexico*, pages 40–41. Springer-Verlag, 2008.

List of Figures

List of Tables