FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Towards an Alternative Approach for Combining Ontology Matchers

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering / Internet Computing

eingereicht von

Simon Steyskal
Matrikelnummer 0828067

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Priv.-Doz. Dipl.-Ing. Dr. Axel Polleres

Wien, 14.09.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Towards an Alternative Approach for Combining Ontology Matchers

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering / Internet Computing

by

Simon Steyskal
Registration Number 0828067

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.-Doz. Dipl.-Ing. Dr. Axel Polleres

Vienna, 14.09.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Simon Steyskal
Anton Baumgartnerstraße 44 B3/106, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

For the work of this master's thesis I had the pleasure to be employed at Siemens AG - Corporate Technology as research scientist. I would like to thank Axel Polleres, my supervisor, for making this possible and for always motivating me during my work. Thanks to the whole Siemens CT RTC BAM CON-AT staff for a wonderful time at Siemens and especially to Herwig Schreiner and Gottfried Schenner for many interesting discussions and feedback regarding my thesis.

Of course I also want to thank all of my friends which supported me during my study and gave me a great time over the last years and my father Peter, who financially supported me during this time.

Furthermore I want to thank my mother Siegrid and sister Sarah for their believe in me, their continuous support and for the uncountable hours of dog sitting they spent, without my study would not have been possible at all.

Finally I want to mention my two dogs Faby and Pino which are two of the most important things of my life.

Abstract

The existence of a standardized ontology alignment format promoted by the Ontology Alignment Evaluation Initiative (OAEI) potentially enables different ontology matchers to be combined and used together, exploiting their respective strengths in a combined matching approach. In the present thesis, we present a novel architecture for combining off-the-shelf ontology matchers based on iterative calls and exchanging information in the form of reference alignments. Unfortunately though, only a few of the matchers contesting in the past years' OAEI campaigns actually allow the provision of reference alignments in the standard OAEI alignment format to support such a combined mapping process. We bypass this lacking functionality by introducing an alternative approach for aligning results of different ontology matchers using simple URI replacement to "emulate" reference alignments in the aligned ontologies. While some matchers still consider classes and properties in ontologies aligned in such fashion as different, we experimentally prove that our iterative approach benefits from this emulation, achieving the best results in terms of F-measure on parts of the OAEI benchmark suite, compared to the single results of the competing matchers as well as their combined results. The new combined matcher – Mix'n'Match – integrates different matchers in a multi-threaded architecture and provides an anytime behavior in the sense that it can be stopped anytime with the best combined mappings found so far.

Kurzfassung

Das standardisierte Ontology Alignment Format der Ontology Alignment Evaluation Initiative (OAEI) erlaubt es, verschiedene Ontology Matcher zu kombinieren, um deren Stärken auszunutzen und Schwächen kompensieren zu können.

In der vorliegenden Diplomarbeit präsentieren wir einen neuen Ansatz, um verschiedene, existierende Ontology Matcher zu kombinieren. Dafür werden iterative Aufrufe dieser Matcher getätigt und zusätzlich Wissen in Form von Reference Alignments ausgetauscht. Da jedoch nur eine kleine Teilmenge der verfügbaren Matcher Reference Alignments direkt untersfützt, stellen wir einen alternativen Ansatz für den Import und die Verwendung von zusätzlichem Wissen in Form von Reference Alignments vor: Basierend auf der Ersetzung von URIs von Entitäten, für welche ein Alignment gefunden wurde, ist es möglich Reference Alignments zu 'emulieren'. Obwohl einige Matcher Entitäten mit der selben URI nicht als vollkommen ident betrachten, zeigen wir empirisch, dass unser interativer Ansatz der Matcher-Ausführung dennoch von solchen Ersetzungen profitiert, wobei wir in Teilen der OAEI Benchmark Tracks die besten Resultate im Vegleich mit bestehenden Ontologie-mapping Tools erreichen.

Der resultierende neue Ontology Matcher *Mix'n'Match* integriert verschiedene Ontology Matcher in einer multi-threaded Architektur und bietet zudem die Möglichkeit den mappingprozess jederzeit - mit den besten zurzeit gefundenen Ergebnissen - zu beenden.

Widmung

Für meinen Großvater, Karl Weichinger, welcher den Abschluss meines Studiums leider nicht mehr miterleben durfte.

Opa, ich möchte dir auf diesem Wege ganz besonders dafür danken, dass du in allen Lebenslagen zu mir gehalten und mir im Laufe der Jahre unsagbar viel beigebracht hast. Du warst für uns alle eine Bereicherung und hast uns immer gezeigt, dass man seine Familie wertschätzen soll und sie viel wichtiger als alles Materielle auf dieser Welt ist. Ich weiß, du wärst sicher sehr stolz auf uns gewesen und obwohl du bei meinem Abschluss nun nicht mehr dabei sein kannst, bin ich mir sicher, dass du - wo auch immer du gerade bist - über uns wachst.

DANKE

Contents

List of Figures

List of Tables

CHAPTER 1 Introduction

1.1 Motivation

Mapping between the internal models of software systems is a complicated and tedious task in almost all data and system integration scenarios, keeping computer scientists busy for the last decades with still no "silver bullet" in sight.

Application domains range from:

database integration: classical database integration scenarios for (mostly) relational databases, where particular problems involve translating and transferring data [24] in migrating or merging between different data bases and merging several database schemata into a global schema [3].

P2P communication: use cases enabling agent and service communication on the Web or in P2P networks [33] (e.g. for mediator systems)

schema mapping: mapping of object-oriented models from different software systems (on an abstract level, this use case is notably also mentioned in [68] already)

Particularly, the latter use case is important for infrastructure providers like Siemens; throughout the life-cycle of products (ranging from e.g. engines, turbines, to trains or complex health-care equipment) we use various software tools related to configuration such as sales configurators, product configurators used during production/manufacturing and separate tools again for software configuration of electronic parts in end customer deployments, each tool with its own underlying internal model. These tools use internal, partially overlapping representations (mostly object-oriented models) of the final products and its parts, which need to be transferred from one configuration step to the next, thus requiring alignment of the underlying models.

In quest for the said "silver bullet" [34] to solve such alignment problems, ontologies and ontology mapping [12, 31, 50] seem to be a good starting point particularly for aligning object-oriented models, since ontologies can represent (and be extracted from) object-oriented models fairly naturally, where the hope is that the increasing number of off-the-shelf ontology mapping tools can be used "out of the box" to propose reference alignments that can be tailored in a semi-automatic process.

1.2 Problem Statement

In the past decade, the field of ontology mapping has matured with many different ontology matchers having participated in the last years' OAEI [1] campaigns, exposing different strengths and weaknesses. The different tracks provided by the OAEI target different mapping problems and since every mapping tool has its specific special techniques and features, good performance in a specific sub-track often comes in hand with a bad performance in another one. So, intuitively, combining the results of a heterogeneous set of ontology matchers, taking the "best off-the-shelf matcher for the problem at hand" seems to be a promising approach for a "universal matcher" without the need for designing a new matcher from scratch.

Unfortunately, (i) neither deciding which matchers suit best for a set of given ontologies, nor (ii) using ontology matchers in a semi-automatic human-guided mapping process, nor (iii) combining their alignment results efficiently is trivial. For (i), selecting the most suitable matcher(s) requires some preparatory work like interviews or tests [60] or background knowledge in terms of training sets for learning a classifier [27, 28].

On the other hand, for an interactive mapping process, one needs to be able to guide a matcher by providing (or confirming) reference alignments, either provided by a domain expert directly or iteratively added by confirming mapping results from an ontology matcher to be used as reference for supporting another call of the same or another ontology matcher.

1.3 Aim of the Work

Aim of this master thesis is the evaluation of different current state-of-the-art ontology matchers and the development of a proof of concept implementation to evaluate the feasibility of an approach, which should be able to combine off-the-shelf ontology matchers in an iterative manner by reusing gathered knowledge.

Our idea is strongly inspired by the definition of *ontology mapping* shown in Figure 1.1: that is, ontology mapping can be summarized as a process which computes *alignments A′* for a pair of input ontologies *O1* and *O2*. There are three more com-

[1] http://oaei.ontologymapping.org/

ponents which can extend this basic definition of the mapping process: (*i*) *input* or *reference alignments* which support the mapping process with initial knowledge about the mapping domain; (*ii*) resources either some sort of background knowledge, e.g. in terms of dictionaries, and (*iii*) parameters, e.g. for narrowing the results by using specific weights or thresholds.

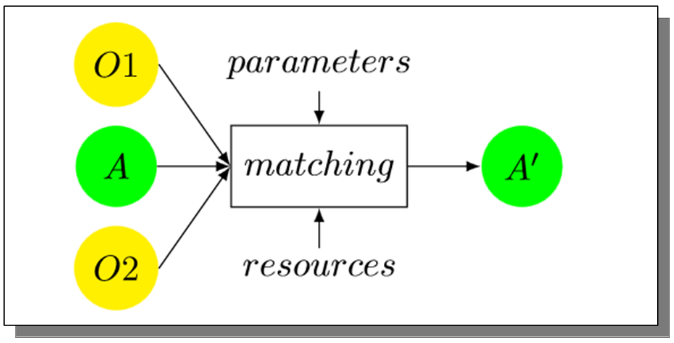


Figure 1.1: Ontology matching process as defined in [30,31]

In this thesis we will particularly focus on the usage of *reference alignments* stemming from (combined) mapping results from different matchers as well as *weighting parameters* (for deciding which combined alignments to approve) to support our iterative mapping process. As for reference alignments, the existence of a standardized ontology alignment format promoted by the Ontology Alignment Evaluation Initiative (OAEI) should enable the usage of off-the shelf ontology matchers in this process that support the said alignment format.

Our hypothesis is that it should be possible to reuse gathered knowledge in an iterative fashion in terms of found alignments and then enrich the to be matched ontologies with this additional knowledge. It should be possible to develop a combined ontology matcher, which performs better on average than a single matcher on a number of heterogeneous ontology mapping problems (from the OAEI campaign), without the need to choose a single matcher, but by iteratively combining results of different matchers, and feeding them as support into subsequent runs of those matchers.

The big advantage of such an approach of altering the ontologies instead of trying to change the mapping process itself, is the flexibility in choosing off-the-shelf ontology matchers for the combined matching approach. They do not have to fulfill specific criteria like the possibility to use background knowledge or parameters for the mapping process.

1.4 Structure of the Work

The present thesis is structured as follows: we start with an introduction into Semantic Web technologies in Chapter 2 and explain the ontology alignment process, alignment formats and common ontology matching techniques in Chapter 3 . Then we continue with an overview about current state-of-the-art ontology matchers which were used for our combined matching approach in Chapter 4 . Our main contribution *the Mix'n'Match approach for combining ontology matchers* is presented in Chapter 5. Chapter 6 contains detailed evaluation results for Mix'n'Match compared to the ontology matchers presented in 4 based on public available datasets. In Chapter 7 we explain the transformation of UML class diagrams into ontologies and present mapping results received by mapping ontologies, which were generated from UML class diagrams. We conclude our thesis and discuss ideas for future improvements and related work in Chapter 8 .

1.5 Impact of this Thesis

A preliminary version of *Mix'n'Match* was published as a short paper in the 12th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2013)[2] [77].
Better evaluation results produced by an improved version of *Mix'n'Match*, which is based on the present thesis, including: (i) a new streamlined architecture by using multi-threading for matcher execution instead of running ontology matchers sequentially in each iteration, as well as (ii) anytime behavior for time consuming mapping tasks not yet tackled in [77] was accepted for the poster session on the Ontology mapping workshop of the 12th International Semantic Web Conference (ISWC 2013)[3].

[2]http://www.onthemove-conferences.org/index.php/odbase13
[3]http://oaei.ontologymapping.org/2013/

CHAPTER 2

Preliminaries

2.1 Ontologies

There exist many similar definitions for the term *ontology* and probably one of the most cited ones was presented by Gruber in 1993:

An ontology is an explicit specification of a conceptualization. [41]

So basically, ontologies are generic conceptual models of a domain of interest. A more formal definition of an ontology is shown underneath:

Definition 1. We consider an ontology as a triple:

$$O = \langle C, I, P \rangle$$

C - Set of Classes Classes or concepts are abstract representations of objects. They can be subsumed by other classes and inherit their properties. Furthermore - if not stated otherwise - a class can inherit from more than one superclass.

I - Set of Individuals Individuals are specific representations of objects and usually describe a very concrete type of concepts. The choice whether an object should be modeled as individual or as a class is often not very easy to make and heavily relies on the modeling domain. For example an object `Integer` can either be considered as a subclass of the concept `Datatype` having an individual called „`1`“, or modeled as as individual of the concept `Datatype` in the absence of more specific objects like „`1`“.

P - Set of Properties P contains all properties which define data values for specific attributes (names, ids, . . .) and relations, which describe possibilities to relate entities in ontologies with each other (subclass, equivalence relations, . . .).

In contrast to other structures which aim for storing data in a defined way, like relational databases, ontologies enable the possibility to store semantics of data, together with specific rules which describe the schema. The possibility to infer new knowledge based on the available data as well as to be able to detect semantic conflicts between the entities of an ontology are additional benefits for using ontologies over common databases to represent and store data.

To understand the principles behind the Semantic Web, some of its major technologies and standards are described in the following chapter and some of them are represented in the *Semantic Web Stack* in Figure 2.1.

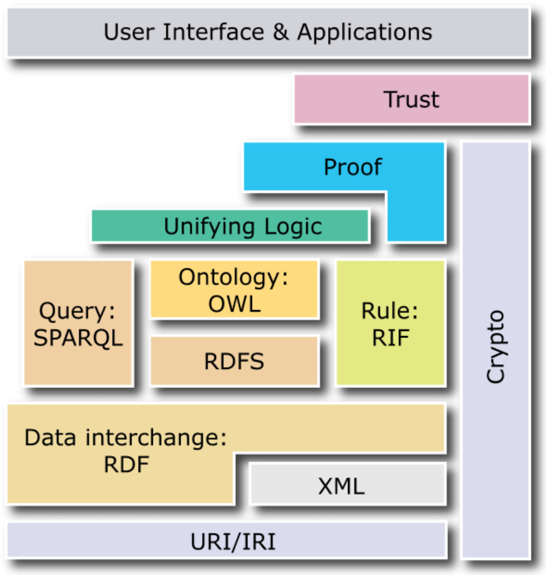


Figure 2.1: The Semantic Web stack [9]

Built upon the *URI/IRI* layer, all higher layers in the *Semantic Web Stack* can uniquely identify their defined resources by URIs (Uniform Resource Identifier) and IRIs (Internationalized Resource Identifier) which are common resource identifiers in the World Wide Web.

The next layers are XML (eXtensible Markup Language) and RDF (Resource Description Framework), which describe the basic language of the Semantic Web and using RDF, which is based on the XML format, enables the possibility to describe resources both in a human-readable and machine-processable way, which will be described together with its most common formats in Section 2.2.1.2.

There currently exist two extensions for RDF, namely *(i) RDF Schema 2.3* and *(ii) the Web Ontology Language 2.4.2.4*. Only by the use of these extensions it is possible to define and model ontologies, since RDF does not provide the possibilities for describing properties or complex relations between resources [31].

In the following sections, we will describe selected parts of the Semantic Web stack in more detail.

2.2 Resource Description Framework (RDF)

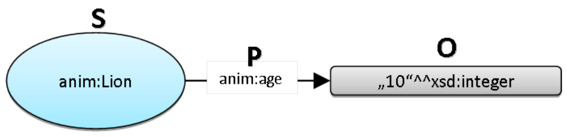


Figure 2.2: Simple RDF graph

The Resource Description Framework (RDF) [45] became a W3C recommendation in 1999 [53] and was revised several times until it became its last W3C recommendation in 2004 [4]. It is a framework for describing and representing information about resources in the World Wide Web and is both human-readable and machine-processable, which enables the possibility to easily exchange information among different applications using RDF triples, but still be easy to read.

In RDF everything is a resource, uniquely identified by its URI and all data is represented as (subject - predicate - object) triples, where subjects and predicates are URIs and objects can either be literals (strings, integers, ...) or URIs as shown in Figure 2.2.

Furthermore subjects or objects can be represented using *blank nodes*, those *blank nodes* do not have a corresponding URI which could identify them and are usually used to express anonymous resources (e.g. »Pino has a friend who is 24 years old« where a *blank node* would represent the anonymous friend of Pino.)

Since one RDF triple usually does not describe a resource entirely, more triples are defined and combined in an *RDF Graph*. Such an *RDF Graph* connects those triples by a simple AND operator and can therefore easily be merged with other *RDF Graphs* without losing entailment information, relying on RDFs monotonicity of semantic extensions [45].

2.2.1 RDF Serialization Formats

There exist several formats for representing and serializing RDF data such as *Turtle(N3)* [5, 6] and *RDF/XML* [4]. In this section we will briefly explain each format and give an example serialization of a small sample triple set which is depicted in Figure 2.3.

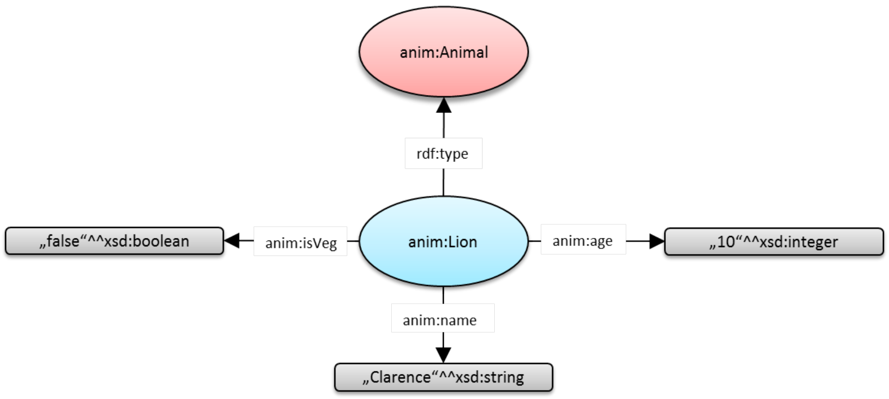


Figure 2.3: An RDF graph describing an animal domain.

subject	predicate	object
anim:Lion	rdf:type	anim:Animal
anim:Lion	anim:age	10
anim:Lion	anim:name	Clarence
anim:Lion	anim:isVeg	false

Table 2.1: The RDF triples encoded within the RDF graph shown in Figure 2.3

2.2.1.1 Turtle and N3

The *Terse RDF Triple Language*, or *Turtle*, is a very lightweight and easy readable subset of the *Notation3* serialization format for RDF and became a W3C Candidate Recommendation in February 2013.[1].

It is commonly used for representing ontologies since it perfectly illustrates the nature of RDF to model data as triples. The simplest statement using *Turtle* consists of a `subject`, `predicate` and `object` which are separated using whitespaces and terminated by a dot. Furthermore it is possible to omit the leading subject, if several triples only vary in their predicates and objects but have the same subject, by terminating each triple(but not the last one) with a semicolon instead of a dot. Listing 2.1 shows the representation of a sample ontology using *Turtle*.

All examples within this thesis are serialized using the *Turtle* format.

Listing 2.1: Turtle representation of the RDF graph in Figure 2.3

```
@prefix : <http://ontology.org/onto1.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix anim: <http://www.example.org/animal_onto#> .
@base <http://www.w3.org/2002/07/owl#> .
```

[1]http://www.w3.org/TR/turtle/

```
<http://www.example.org/animal_onto/Lion>
        rdf:type <http://www.example.org/animal_onto/Animal>;
        anim:name "Clarence" ;
        anim:isVeg "false" ;
        anim:age "10" .
```

2.2.1.2 RDF/XML

RDF/XML is the most common format for representing ontologies and natively supported by all RDF parsers. Unlike *Turtle* it is a XML-based serialization of RDF, which inevitably leads to a larger overhead when representing RDF triples. It was introduced together with the RDF specification in 1999 and became a W3C Recommendation in February 2004.[2] As shown in Listing 2.2, RDF/XML serializations tend to be verbose and more difficulty readable by humans. An approach to make *RDF/XML* more concise was proposed by Brickley in 2002 and is called „Striped RDF/XML Syntax“ [10]. The striped *RDF/XML* syntax introduces XML elements for nodes and arcs of an RDF graph and provides the possibility to group triples as shown in Listing 2.3.

Listing 2.2: RDF/XML description of the RDF graph in Figure 2.3

```
<?xml version="1.0"?>

<rdf:RDF xmlns="http://ontology.org/onto1.owl#"
      xml:base="http://ontology.org/onto1.owl"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:anim="http://www.example.org/animal_onto#">

    <rdf:Description rdf:about="http://www.example.org/
        animal_onto/Lion">
        <rdf:type rdf:resource="http://www.example.org/
            animal_onto/Animal"/>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.example.org/
        animal_onto/Lion">
        <anim:name>Clarence</anim:name>
    </rdf:Description>
        ....
</rdf:RDF>
```

[2]http://www.w3.org/TR/REC-rdf-syntax/

Listing 2.3: Striped RDF/XML description of the RDF graph in Figure 2.3

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://ontology.org/onto1.owl#"
     xml:base="http://ontology.org/onto1.owl"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:anim="http://www.example.org/animal_onto#">

    <anim:Animal rdf:about="http://www.example.org/animal_onto/
       Lion">
      <anim:name>Clarence</anim:name>
      <anim:age>10</anim:age>
      <anim:isVeg>false</anim:isVeg>
    </anim:Animal>
</rdf:RDF>
```

2.3 RDF Schema (RDFS)

Although RDF provides the basic elements and tools for describing web resources, it does not offer the possibility to describe relations or constraints between entities and therefore is not able to describe ontologies. This lack of functionality included the development of *RDF Schema (RDFS)*, which is a semantic extension to the basic RDF specification and provides the capability to describe properties and relations among resources and therefore offers basic elements for ontology description.

RDFS was firstly published in 1998 and became a W3C recommendation in 2004 [11].

RDFS now divides resources into two groups:

Classes: The first group of resources is called classes. Those classes are usually identified by URIs and described using RDF properties, a member of a specific class is called its instance, which is denoted by the rdf:type property. A class can have a set of instances of itself, which is called its class extension. Furthermore classes can share the same set of instances although they might be different classes (e.g. Alice defines dogs as animals and Bob defines them as carnivores, it is possible for those two classes to have the same instances but of course, different properties).

RDFS introduces subclass relations among classes, namely if there exists a class A which is a subclass of a class B, then all instances of A will also be instances of B and vice versa, if a class B is a superclass of a class A, then all instances of A are also instances of B. The `rdfs:subClassOf` property may be used to represent this subclass relation.

A small sample ontology using *RDFS* features and describes a teacher/pupil domain is shown in Figure 2.4.

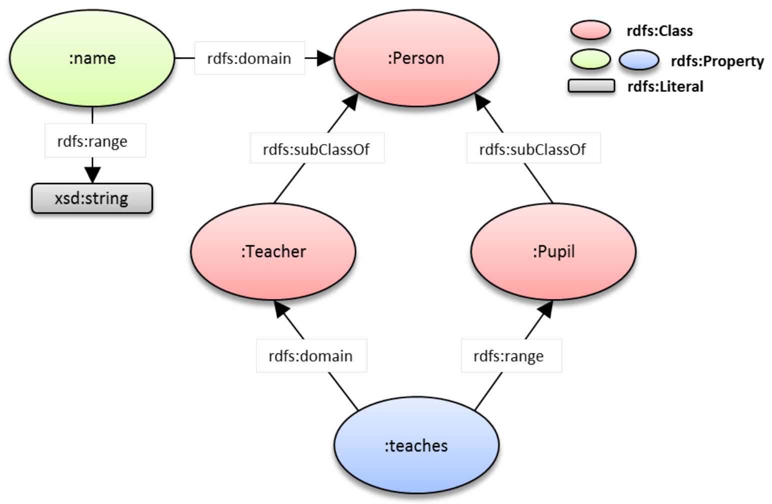


Figure 2.4: Small sample ontology using RDFS features

Important Classes:

rdfs:Resource: Everything described in RDF is a resource and instance of the class `rdfs:Resource`. `rdfs:Resource` is an instance of `rdfs:Class` and all other classes are its subclasses.

rdfs:Class: `rdfs:Class` is an instance of `rdfs:Class` and is the class of resources that are RDF classes. (cf. `:Teacher`)

rdfs:Literal: As mentioned earlier, an object of an RDF triple might be a literal. Those literals are instances of the class `rdfs:Literal` and divided into typed literals, which are instances of its respective datatype class, and plain literals.

rdfs:Datatype: This class describes the class of datatypes and all instances of it are related to a datatype described in the RDF Concepts specification citerdf2. It is both an instance and a subclass of `rdfs:Class` and each instance of `rdfs:Datatype` is a subclass of the `rdfs:Literal`.

rdf:Property: `rdf:Property` is the class of RDF properties and an instance of `rdfs:Class`. (cf. `:teaches`)

Properties: The second group of resources are properties which are defined by [52] as relations between subject resources and object resources.

Like the subclass relation, *RDFS* also introduces the concept of subproperties. If a property A is a subproperty of a property B, then all resources which are connected by A are also connected by B and vice versa. This subproperty relation indicated by the `rdfs:subPropertyOf` property.

Important Properties:

rdfs:domain This property states that any resource which has a given property must be an instance of the class referenced by `rdfs:domain`. (cf. `:teaches` and `:Teacher`)

rdfs:range `rdfs:range` is used to state that the values of a given property are instances of the class referenced by `rdfs:range`. (cf. `:teaches` and `:Pupil`)

rdf:type An important property which states that a resource is an instance of a class.

rdfs:subClassOf & rdfs:subPropertyOf As mentioned above, these properties are used to state the subclass and subproperty relations among classes and properties. Both are instances of `rdf:Property`. (cf. `:Teacher` and `:Pupil` are subclasses of `:Person`)

rdfs:label & rdfs:comment These properties are instances of `rdf:Property` and may be used to provide a human-readable description of the resource itself as well as its name.

2.4 Web Ontology Language (OWL)

Although *RDFS* allows the representation of simple ontologies by using properties, which describe the hierarchical relation among entities, it lacks in the support of defining more sophisticated entity relations (e.g. disjointness), cardinality (e.g. exactly one), equality (e.g. equivalences between classes/properties/instances) and characteristics of properties (e.g. symmetry). For that purpose the *Web Ontology Language (OWL)*, which was firstly published in 2002 and became a W3C recommendation in 2004 [57], was developed. Since 2012 an extension to *OWL*, called *OWL 2*, is available as W3C recommendation [40].

In general *OWL* is used to describe complex ontologies and furthermore introduce the possibility to automatically process the content in the given ontology by using the previous mentioned constructs which were not available in *RDFS*.

2.4.1 OWL Sub-languages

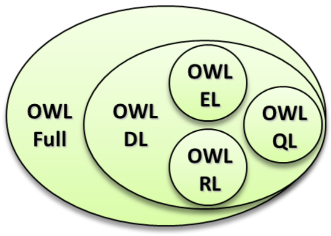


Figure 2.5: The OWL sub-language hierarchy as indicated in [57]

In order to sufficiently fulfill the different requirements of ontologies and especially avoid an unnecessary increase of complexity of those ontologies, three different sub-languages of *OWL* were developed, namely: *OWL Lite*, *OWL DL* and *OWL Full*. Each of these sub-languages is a subset of the more complex one as indicated in Figure 2.5. As a result, following validity conclusions hold [57]:

OWL EL/QL/RL: These three profiles introduce restrictions on *OWL* in order to allow more efficient reasoning. *OWL EL* provides the expressiveness of large-scale ontologies but only needs polynomial time for selected reasoning problems such as classification and instance checking. *OWL QL* is used to implement sound and complete query answering on top of relational databases and *OWL RL* provides the possibility to run rule-based reasoning algorithms in polynomial time.

OWL DL: Increasing the expressiveness of *OWL EL/QL/RL* but still be computational complete and decidable, leads to *OWL DL*, which is translatable into the expressive Description Logic *SROIQ* [2] . Although it includes all language concepts of *OWL*, they can only be used under special conditions (e.g. a class cannot be an instance of another class, but of course be its subclass).

OWL Full: Losing the restriction of using *OWL* language constructs only under certain conditions and therefore retrieving the most expressiveness and syntactic freedom for defining OWL ontologies, unfortunately comes in hand with the loss of computational guarantees. As indicated in [57] it is very unlikely, that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Remark: Although every ontology expressed in *OWL* is a valid *RDF* document, not every *RDF* document is a valid *OWL* ontology. Only *OWL Full* is a complete extension of *RDF*, whereas *OWL DL* and the profiles *OWL EL/QL/RL* are restricted extensions of *RDF*. When migrating from an *RDF* document to an *OWL DL/EL/QL/RL* ontology, those restrictions must be fulfilled.

2.4.2 OWL Features

OWL introduces many new features for describing information and knowledge about a domain and is even more expressive than *RDFS*. We will now introduce some of this features in more detail based on a sample ontology, illustrated in Figure 2.6.[3]

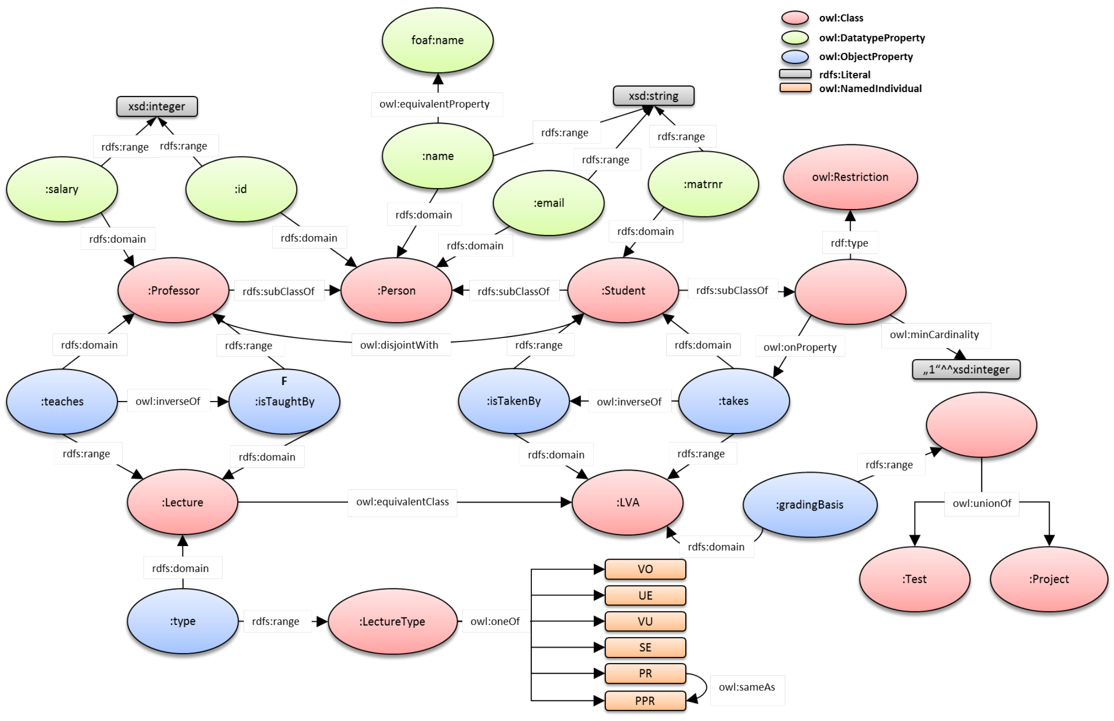


Figure 2.6: Sample ontology, which uses selected OWL features

2.4.2.1 Properties

owl:DatatypeProperty Datatype properties link individuals to data values. (cf. `:id`, `:email`)

owl:ObjectProperty Object properties are used to define relations between classes (i.e. link individuals to individuals). (cf. `:isTakenBy`, `:takes`)

owl:FunctionalProperty If a property is defined as `owl:FunctionalProperty` it can have only one unique value for each instance of this property. (cf. `:isTaughtBy`; a `:Lecture` can only be held by one `:Professor`)

[3]Note that the following list is only a small subset of the features *OWL* provides and only contains those features of *OWL*, which we use in the present thesis.

2.4.2.2 Relations between Entities

owl:equivalentClass This property is used to define the similarity between two classes. (cf. `:Lecture` and `:LVA`)

owl:equivalentProperty This property is used to define the similarity between two properties. (cf. `:name` is equivalent to the property `foaf:name`, defined in the FOAF ontology[4])

owl:sameAs This property is used to define the similarity between two instances. (cf. the two lecture types `:PR` and `:PPR` are equivalent)

owl:inverseOf Using `owl:inverseOf` offers the possibility to state an inverse similarity between two properties. (cf. `:isTakenBy` and `:takes`)

owl:disjointWith Using `owl:disjointWith` offers the possibility to state that two entities are disjoint. (cf. an instantiation of `:Professor` cannot be a `:Student` too)

2.4.2.3 Boolean Connectives and Enumeration

owl:unionOf This property links a class to a union of class descriptions. (cf. *:gradingBasis*, its `rdfs:range` is a blank node, which is defined as union of `:Test` and `:Project`)

owl:oneOf This property is used to define enumerations within ontologies. (cf. `:LectureType` contains one of the listed individuals)

2.4.2.4 Restrictions

owl:Restriction Restrictions are subclasses of `owl:Class` and are used to define value constraints for specific properties. (cf. a `:Student` must take at least one `:Lecture`)

owl:onProperty The `owl:onProperty` property defines the specific property for which the restriction holds. (the above mentioned restriction is defined for the property `:takes`)

owl:(min | max)Cardinality One example of a restriction mentioned above are cardinality constraints. Whereas `owl:minCardinality` and `owl:maxCardinality` define lower and upper bounds for cardinalities, the `owl:Cardinality` property is used to define a precise value for the cardinality. (`owl:minCardinality 1` is used to state, that a `:Student` must take at least one `:Lecture`)

[4]http://xmlns.com/foaf/spec/

2.5 Unified Modeling Language (UML)

The Unified Modeling Language [35] is a modeling language, which defines a (*i*) notation for graphically representing software systems and (*ii*) a meta-model which defines this notation. It is a permanently evolving standard and currently available in Version 2.4.1[5], which was released in August 2011 by the Object Management Group (OMG)[6].

As mentioned above, UML has its prime use-case in representing software systems, especially those originated from object-oriented programming and due the large variety of different diagram types provided by UML, which are mainly divided into structural and behavioral ones as shown in Figure 2.7, UML is capable of specifying all aspects of systems with respective diagrams.

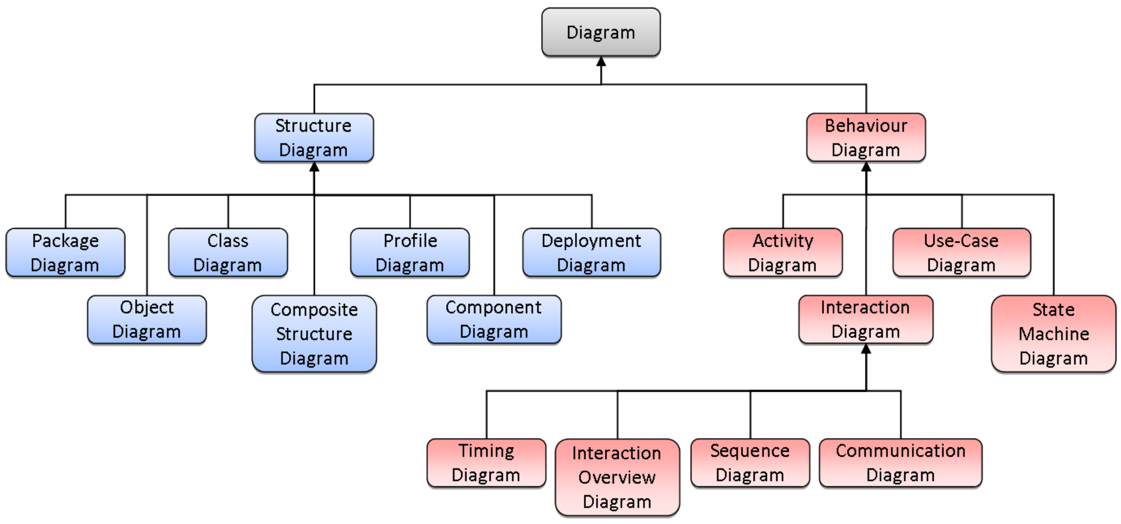


Figure 2.7: Diagrams of UML divided into structural and behavioral diagrams [39]

For the topic of the present thesis, especially the class and object diagrams are of particular interest and will be described in detail in the following subsections.

2.5.1 Class Diagram

When using object-oriented methods for designing and developing software systems, the usage of class diagrams has become somehow mandatory in order to be able to describe the various relationships and objects within such a system appropriately. A sample UML class diagram is depicted in Figure 2.8 and consists of following elements:

[5]http://www.omg.org/spec/UML/2.4.1/
[6]http://www.omg.org/

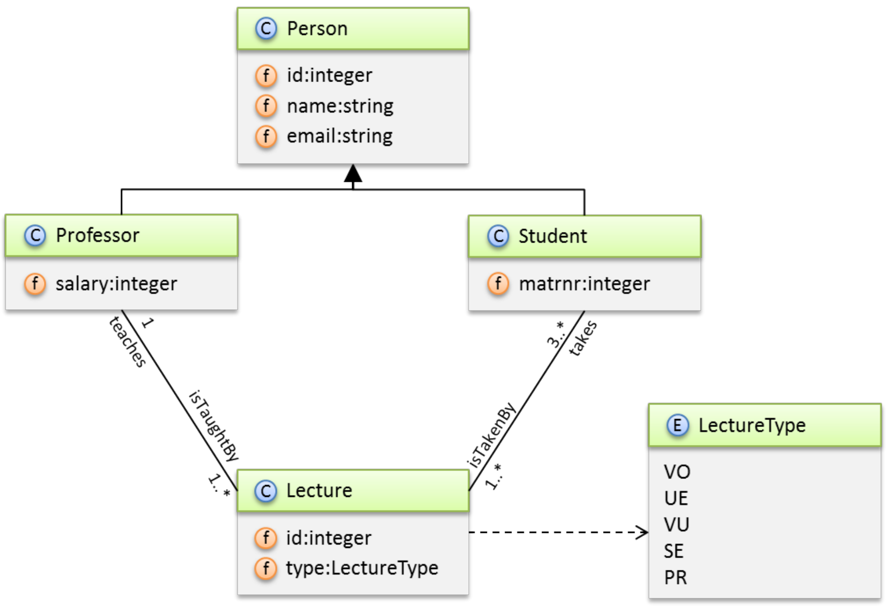


Figure 2.8: A sample UML class diagram

Class: The most important element of class diagrams is the class itself, which describes objects together with their abilities. Its respective notation is a rectangle, split into 3 sections, which contain the name of the class, its attributes and its methods. (cf. *Person* and *Student*)

Association: Associations represent bidirectional relationships between instances of classes or generally speaking the conceptual relationships between classes. Furthermore each association has two roles, depending on the direction of the association, which can be explicitly named by a label. Multiplicities which are added to every role define how many objects are involved in the given relationship and are usually represented as pair of *min* and *max* values. (cf. *takes* and *isTakenBy*)

Attribute: As already mentioned in the description of classes, attributes are properties of classes. They have to consist of a name and a type and can be extended by default values, visibility definitions and multiplicities which indicate the occurrences of those attributes within classes, similar to the multiplicities used for associations (default is 1). (cf. *matr:integer* and *name:string*)

Generalization: The concept of generalization defines the inheritance of properties from one class to another class. Such an generalization relationship is represented as a line between one source class (specialization class) to the target class (general class), having an empty arrowhead on the target side. Unless otherwise indicated, all attributes and methods of the target class are available in the source class but not vice versa. (cf. *Person* and *Student, Professor*)

Aggregation: As a specific type of association, an aggregation represents a part-of relationship between two classes. Its notation is a line between a source and a target class, having an empty diamond on the source side, which indicates that the source class is part of the target class. As mentioned in [35] there exists no single accepted definition of the difference between aggregations and associations, which makes it difficult to decide whether an association should be modeled as aggregation or not.

Composition: Similar to aggregations, compositions define a part-of relationship between classes but in addition declare the respective target class as owner of the source class. This means that even if a class is part of many other classes, its instances can only have one owner. For example, a PC can be located in any room, but one specific PC can be located in only one room at a time. Regarding the notation of compositions, it is similar to those of aggregations, but instead of an empty diamond on the source side, a composite relation is modeled with a filled diamond.

Multiple classification: As an extension to single classification, which defines the relation between a class and its object, multiple classification allows the definition of more than one class an object may belongs to, but without defining exactly to which class it belongs. Furthermore it allows the representation of complete sets and disjointness, e.g. every person must be either a man or a female but cannot be none or both of them.

Association class: In order to be able to define additional properties for relationships between classes, association classes are used and added to their respective association.

Other elements: Note that we only explained the most common and used elements of UML class diagrams. There are many other elements of UML class diagrams such as constraints, abstract classes, parametrized classes, etc. which also help to describe the behavior and implementation of classes within an object-oriented environment.

2.5.2 Object Diagram

A UML object diagram is a concrete representation of its related class diagram, where all attributes have specific values and all relations between classes are represented as relations between their objects. Since object diagrams serve as an instantiation of their respective class diagram, all multiplicities of relations are omitted and replaced by concrete implementations of the respective relationships.

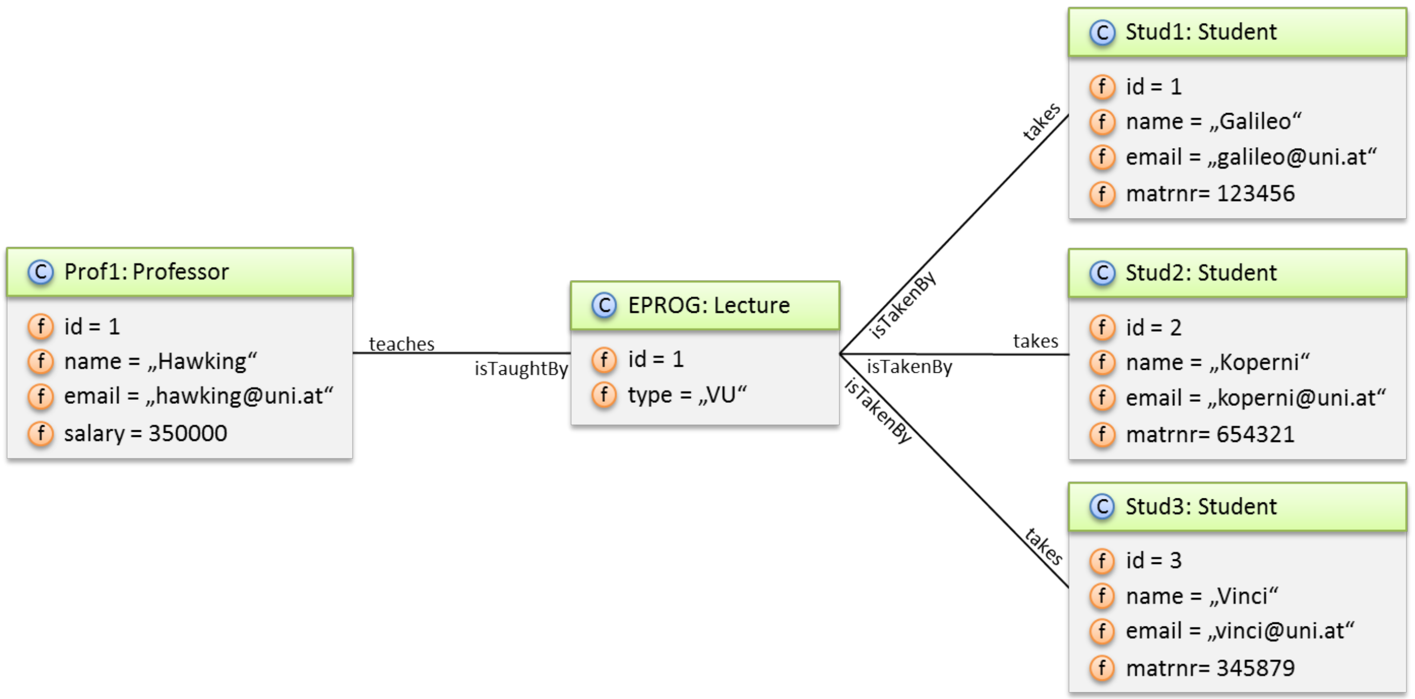


Figure 2.9: A UML object diagram as instantiation of a UML class diagram

CHAPTER 3

Ontology Alignment

As already discussed in Section 1.1, ontology mapping or alignment is a very promising approach for finding similarities and mappings between different data models. Finding such similarities is crucial for integration tasks since heterogeneity of data models cannot be avoided [31].

Heterogeneity is primarily caused by the fact that different modeling experts have different ways of expressing their knowledge, use different tools for creating their models and often work on different levels of detail.

In this chapter we will first discuss the ontology alignment process in more detail, afterwards we introduce the standard OAEI format to represent mappings between entities in ontologies and then explain some common matching techniques used by existing matchers.

3.1 The Alignment Process

In order to define ontology alignment, or more precisely the ontology alignment process, we rely on the definition presented in [30, 31]: basically ontology alignment or ontology mapping can be summarized as a process which computes a set of *alignments* A' for a pair of input ontologies *O1* and *O2*. A' consists of alignments A, which define mappings between entities defined in the matched ontologies and can have various types of cardinalities, e.g. 1:1, 1:n, n:m. Extending the definition presented in [75]:

Definition 2. We define an alignment A as a quintuple

$$A = \langle id, e1, e2, r, c \rangle$$

such that:

id represents an unique identifier for the respective mapping

e1,e2 where $e1, e2 \in (C \cup I \cup P \cup R)$; represent the aligned entities in both ontologies

r is the relation of the alignment, which can either be equivalence($=$), generalization ($\sqsupseteq$), subsumption and supersumption ($\sqsubseteq$, $\sqsupseteq$) or disjointness ($\perp$) [1]

c where $c \in [0, 1]$; this value represents the confidence of the respective matcher in its found alignment. The higher the value, the higher the matchers trust in its found alignment.

As shown in Figure 1.1, there are three more components which can extend this basic definition of the mapping process:
(*i*) **input** or **reference alignments** which support the mapping process with initial knowledge about the mapping domain and are usually provided by a human domain expert;
(*ii*) **resources** either some sort of background knowledge, e.g. in terms of dictionaries, and
(*iii*) **parameters**, e.g. for narrowing the results by using specific weights, thresholds or values for used matching techniques.

3.2 Alignment Format

Ontology alignment is usually used to find mappings and mappings between entities of ontologies. In order to be able to use such alignments in a wider context, e.g. for creating an integrated data model combining information and knowledge used in both ontologies, using a standardized alignment format is necessary.

For this purpose, several alignment formats for representing mappings and making them interchangeable between different tools were proposed over the last years such as *SBO* [55,76], *SWRL* [47], *SKOS* [42,58], *OWL* [81] and the *OAEI Alignment Format* [29, 33].

In this section we will focus on the alignment format proposed by the OAEI and explain its structure and capabilities in more detail.

3.2.1 OAEI Alignment Format

As mentioned above, the alignment format proposed by the OAEI is one of the most common formats for representing alignments between ontologies. It aims at being a simple, easy producible and extensive alignment representation, which is capable of handling complex alignment definitions. Furthermore it supports additional mapping relations different to equivalence.

[1]How to translate this relations in RDF, is shown in Section 3.2

An alignment file consists of a description of metadata about the included mappings, followed by the mappings itself and is encoded using the striped RDF/XML syntax [10].

Listing 3.1 represents a sample alignment file in the RDF representation of OAEI alignment format, which consists exemplarily of three mappings between two sample ontologies.

Listing 3.1: Sample alignment file

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns="http://example.org/alignment"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:onto1="http://ontologymatch.org/onto1.rdf"
        xmlns:onto2="http://ontologymatch.org/onto2.rdf"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>11</type>
  <onto1>
    <Ontology rdf:about="http://ontologymatch.org/onto1.rdf" />
  </onto1>
  <onto2>
    <Ontology rdf:about="http://ontologymatch.org/onto2.rdf" />
  </onto2>

  <map>
        <Cell>
           <entity1 rdf:resource="onto1#TechRepo"/>
           <entity2 rdf:resource="onto2#TechnicalReport"/>
           <measure rdf:datatype="xsd:float">0.513</measure>
           <relation>=</relation>
        </Cell>
  </map>
  <map>
        <Cell>
           <entity1 rdf:resource="onto1#PhdThesis"/>
           <entity2 rdf:resource="onto2#PHDThesis"/>
           <measure rdf:datatype="xsd:float">0.763</measure>
           <relation>=</relation>
        </Cell>
  </map>
  <map>
        <Cell>
           <entity1 rdf:resource="onto1#note"/>
           <entity2 rdf:resource="onto2#note"/>
           <measure rdf:datatype="xsd:float">1.0</measure>
           <relation>=</relation>
        </Cell>
  </map>
</Alignment>
</rdf:RDF>
```

As shown in the listing above, an RDF document, which contains information about an alignment set and is described using the OAEI alignment format, consists of a root class *Alignment* and its respective properties.

3.2.1.1 The Alignment Class

In the following, we will describe the important properties of the *Alignment* class in more detail.

level Depends on the expressiveness of contained mappings.

Level 0: In that case, all entities contained in the alignments can be identified by their URIs. The mappings are defined language independently and can contain alignments between classes, properties and individuals, which is exemplified in Listing 3.2. Furthermore complex mappings are allowed, if the complex terms are identified by an URI.

Level 1: Loosing the restrictions from previous alignment level but keeping its language independence, this level allows the mapping of pairs of sets of entities shown in Listing 3.3.

Level 2: The last alignment level allows the definition of more general mappings. Unfortunately this level is no longer language independent, since the complex mapping expressions are not necessarily identified by an URI. Such an example mapping is depicted in Listing 3.4 using OWL as expressing language and describes the fact that a Writer in the second ontology is someone that hasWritten something in the first one. [31].

Listing 3.2: Level 0 mapping

```
<Cell>
  <entity1 rdf:resource="&onto1;#Writer"/>
  <entity2 rdf:resource="&onto2;#Writer"/>
  <measure rdf:datatype="&xsd;float">1.0</measure>
  <relation>=</relation>
</Cell>
```

Listing 3.3: Level 1 mapping

```
<Cell>
  <entity1 rdf:resource="&onto1;#Writer"/>
  <entity2>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="&onto2;#WriterA" />
      <owl:Class rdf:about="&onto2;#WriterB" />
    </owl:unionOf>
  </entity2>
  <measure rdf:datatype="&xsd;float">0.6364</measure>
  <relation>=</relation>
</Cell>
```

Listing 3.4: Level 2 complex mapping

```
<Cell>
  <entity1 rdf:resource="&onto1;#Writer"/>
  <entity2>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&onto2;#hasWritten"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1</owl:
           minCardinality>
    </owl:Restriction>
  </entity2>
  <measure rdf:datatype="&xsd;float">0.575</measure>
  <relation>&lt;</relation>
</Cell>
```

In our approach we currently only cover **level 0** alignments!

type This property defines the possible cardinalities of the alignments. Different to the common notations like 1:1 (one-to-one), 1:n (one-to-many), and n:m (many-to-many), the multiplicities used in the OAEI alignment format state if the mappings are total (+), injective (?), both injective and total (1) or nothing (*), each for both sides of the mappings. That results in a large amount of possible combinations ?:?, ?:1, 1:?, 1:1, ?:+, +:?, 1:+, +:1, +:+, ?:*, *:?, 1:*, *:1, +:*, *:+, *:*.

Definition 3 (total mapping - +)**.** Considering two ontologies O_1 and O_2. Mappings are declared as *total*, if every entity of O_1 can be mapped to at least one entity of O_2.

Definition 4 (injective mapping - ?)**.** Considering two ontologies O_1 and O_2. Mappings are declared as *injective*, if an entity of O_1 can be mapped to at at most one entity of O_2.

Definition 5 (total and injective mapping - 1)**.** Considering two ontologies O_1 and O_2. Mappings are declared as *total and injective*, if every entity of O_1 can be mapped to at most one entity of O_2.

Definition 6 (unknown mapping - *)**.** Considering two ontologies O_1 and O_2. Mappings are declared as *unknown*, if there does not exist any restrictions of mapping entities of O_1 to entities of O_2.

For the sake of simplicity we declare our produced alignment set as **type *:***!

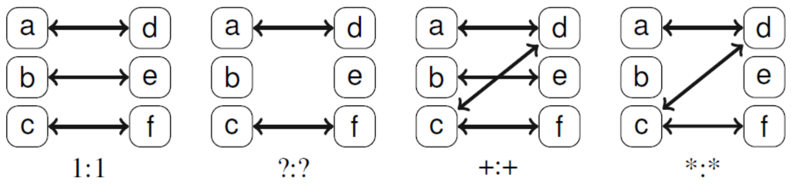


Figure 3.1: Illustration of the different supported arity types [31]

map Map contains information about particular alignments, which are described using the *Cell* class.

3.2.1.2 Cell Class

As mentioned above, the class *Cell* contains information about particular alignments (cf. Listings 3.2 to 3.4).

entity1 Definition or reference to the first matched entity.

entity2 Definition or reference to the second matched entity.

measure Displays the confidence of the matcher which produced the alignments, that this alignment holds and is usually a float value between 0 and 1.

relation Current state-of-the-art ontology matchers only produce equivalence mappings between entities [30] primarily based on the fact that evaluation campaigns like the OAEI only consider those in their reference alignments, since generating a complete set of reference alignments containing all kinds of relations between entities is a tremendous effort and would bias the evaluation results if single matchers does not consider specific relation types. Besides common equivalence relations, the OAEI alignment allows to define all the basic alignment relations mentioned in 2 with the following extensions, disjunction and *HasInstance*, *InstanceOf* [37]. Since this alignment format is based on the Alignment API, all available relations are additionally accessible through a fully qualified class name of the relation implementation (e.g. = is equivalent to fr.inrialpes.exmo.align.impl.rel.EquivRelation).

3.3 Matching Techniques

Since detecting alignments between ontologies manually is a very tedious, ineffective and error-prone task, especially if the ontologies contain a large amount of entities, several different mapping strategies have been explored over the last years. They can basically be divided into four main method categories [31]:

terminological/lexical: Terminological or lexical matching techniques (using for instance string-mapping, string-similarity, or thesauri to match entity names) detect mappings based on similarities in entity names or entity descriptions (e.g. labels, comments). They are one of the most common and most researched matching techniques in terms of schema mapping. Lexical matching techniques perform well on many typical use cases and the majority of ontology mapping tools make use of them since they are mostly easy to implement.

structural: Like terminological techniques, this family of matching techniques is very common in the domain of ontology matchers. They try to align entities, based on their structural similarities (analyzing e.g. the taxonomy graph implied by a subclass hierarchy, etc.).

semantic: We call matching techniques semantic-based if they are using reasoning techniques to interpret complex OWL axioms and prevent logically inconsistent alignments to get accepted. Or if they use other approaches to measure the semantic equality of entities like background information or generic/domain specific rules.

extensional: This category of matching techniques use individuals of concepts to align ontologies. By using the knowledge of already matched properties of sets of individuals, they aim to find possible similarities between concepts.

As addition to the above mentioned techniques, **machine-learning based** Mapping strategies [25,26] can be used to decide whether or not a matching technique performs well on specific datasets. Such strategies train classifiers for deciding whether alignments from specific strategies get accepted or not based on available training sets. The necessity of available and representative training sets is one of the drawbacks of that approach. Among the matchers competing in recent OAEI contests only one state-of-the-art ontology matcher (*YAM*++) uses machine-learning for its mapping tasks [62].

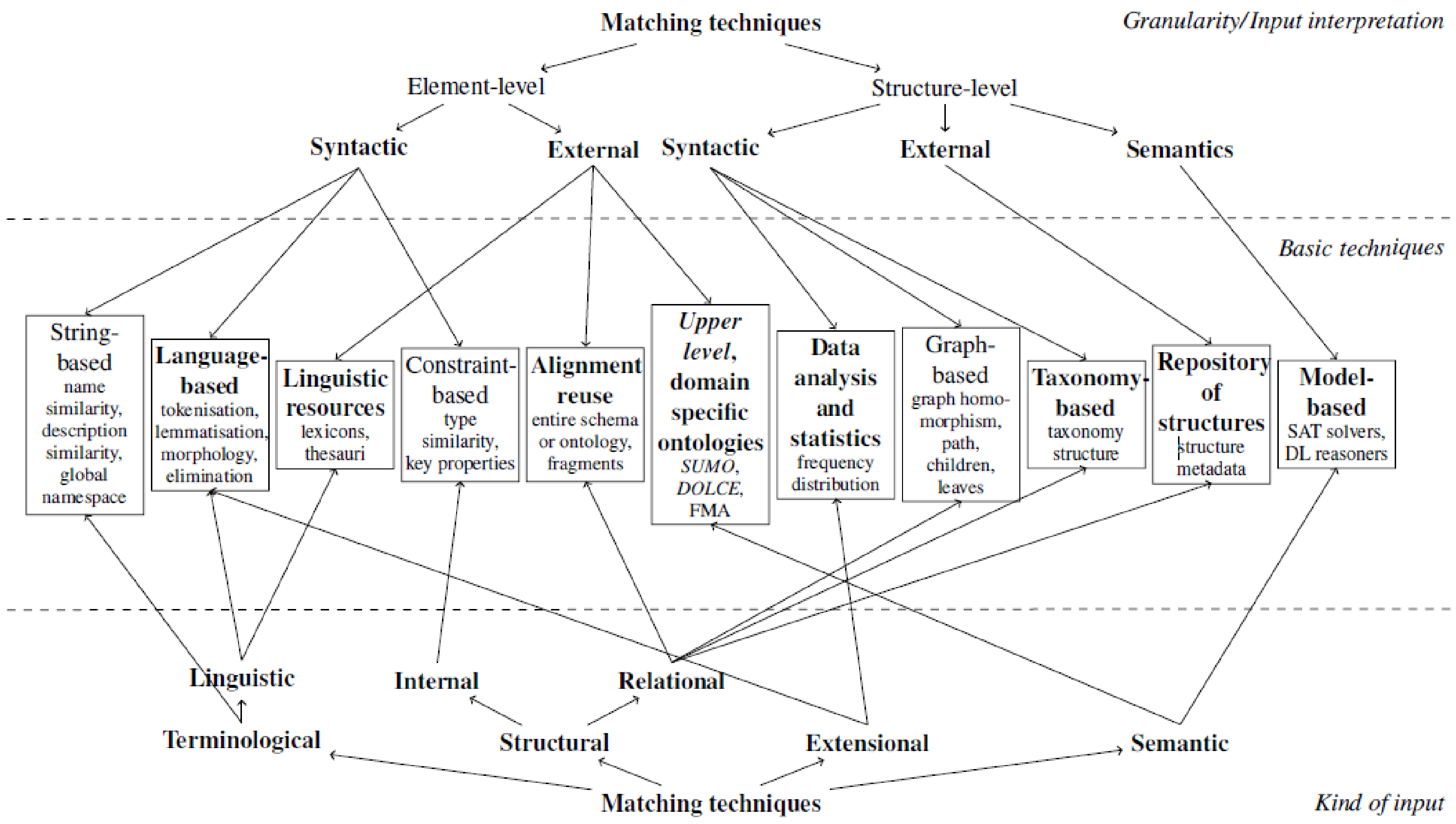


Figure 3.2: Different kinds of matching techniques [31]

In Figure 3.2, a schematic overview of the above mentioned basic matching techniques is illustrated which are broken down into more elementary matching techniques [31] and can be read in a top-down or bottom-up fashion.

top-down: When reading the classification from the top, the techniques are divided by the way of processing the input data. Element-level techniques take entities one-by-one analyzing their attributes and try to find similarities based on individual abilities. Structure-level techniques on the other hand, find mappings based on the relations between entities. Both divide their input data into syntactic techniques which are based on static rules and into external techniques which use e.g. external resources for supporting the mapping process. Structural techniques furthermore can use semantic techniques which try to exploit semantic relations between entities to find similarities.

bottom-up: Starting from the bottom, the techniques are divided into the four basic mapping categories, mentioned before. Table 3.1 illustrates which techniques are used in some state-of-the-art ontology matchers.

Ontology Matcher	terminological based	structure based	semantic based	instance/extensional based
Anchor-Flood	✓	✓	✗	✗
Aroma	✓	✓	✓	✗
AUTOMSv2	✓	✓	✗	✗
Eff2Match	✓	✓	✓	✗
FalconAO	✓	✓	✓	✗
Hertuda	✓	✗	✗	✗
HotMatch	✓	✓	✓	✗
Lily	✓	✓	✓	✗
LogMap2	✓	✓	✓	✓
YAM++	✓	✓	✓	✗

Table 3.1: List of state-of-the-art matchers and their used techniques

3.3.1 Terminological Techniques

The simplest family of matching techniques are the terminological ones. These techniques aim to tackle the terminological heterogeneity among different entities and try to find mappings between them even if they are expressed using different languages or if synonyms, homonyms and acronyms were used.

terminological techniques can be divided into **string-based** techniques which use available lexical information of entities defined in the ontology to match concepts by string comparison using distance metrics like *Jaro-Winkler* [82], *Levenshtein* [54] and

Jaccard; and **language-based** ones which try to improve the quality of terminological matchers by preprocessing the available lexical information using techniques, such as:

- tokenization: breaking a single string up into words (cf. Figure 3.3)

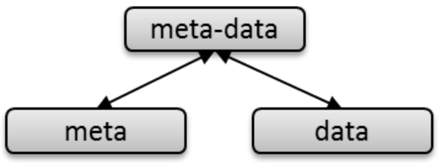


Figure 3.3: Splitting meta-data into meta and data

- lemmatization: reducing different forms of words to a single canonical form (cf. Figure 3.4)

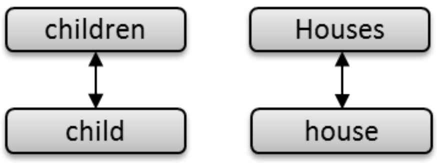


Figure 3.4: Normalizing words into one consistent form

- morphology: uses rules that hold in a certain language to break down complex words (cf. Figure 3.5)

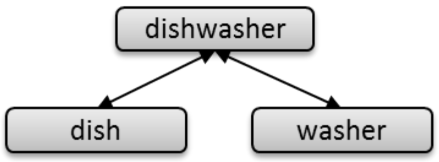


Figure 3.5: Splitting words based on language specific rules

- word elimination: redundant words that add little or no semantics, often called „stop words", are filtered out in the comparison process (cf. Figure 3.6)

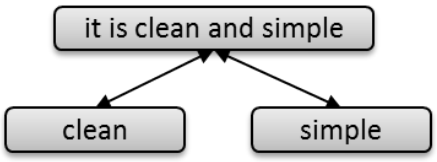


Figure 3.6: Eliminating stop words *it*, *is*, *and*

Unfortunately basic terminological techniques have problems, when dealing with special linguistic characteristics like homonyms, acronyms or synonyms.

homonyms Homonyms are words having the same spelling and/or same pronunciation but different meanings (cf. Figure 3.7)

Figure 3.7: *ant* and *aunt* may have a high string similarity but describe different concepts

acronyms Acronyms are abbreviations of phrases or words, forming a new word (cf. Figure 3.8)

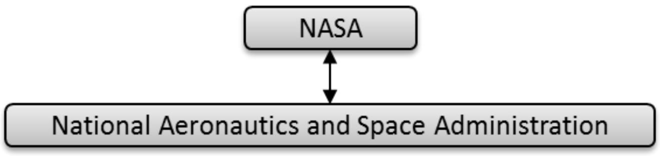


Figure 3.8: The acronym *NASA* and its expanded form have a very small string similarity but the same meaning

synonyms Synonyms are words having the same meaning (cf. Figure 3.9)

strong
solid

Figure 3.9: *strong* and *solid* both describe the same ability

To be able to deal with these phenomena, additional external linguistic resources like WordNet [59] or other thesauri have to be used to support the mapping process.

3.3.2 Structural Techniques

Following the definition of Euzenat [31], **structural** techniques can be split into two types of matching approaches, namely:

internal structure approach: Such an approach uses e.g. the set of properties of classes to find similarities between them. If the datatype, range and/or cardinalities of those properties correspond, it is more likely that the respective attached classes will correspond too as shown in Figure 3.10.

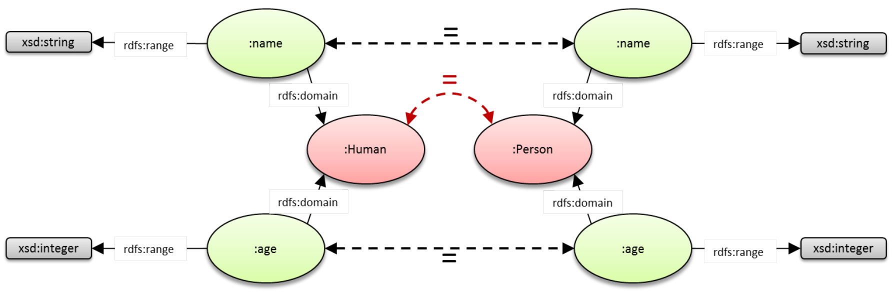


Figure 3.10: The properties of both classes are similar, resulting in a possible mapping between both concepts.

relational structure approach: By representing the ontology as a graph it is possible to find mappings between concepts by comparing their relations, which are represented as edges, to other concepts. Relying on such a relational structure approach, concepts which have a similar relative position in an ontology, e.g. by comparing their *subClassOf* edges, are considered to be similar. An example is shown in Figure 3.11

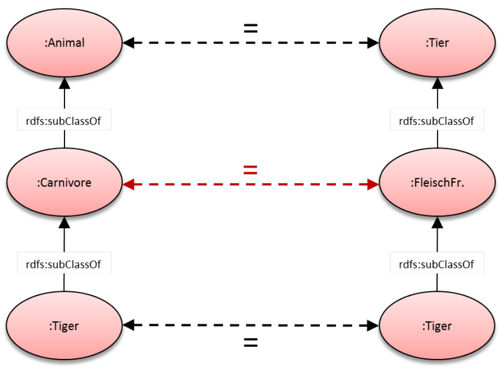


Figure 3.11: The relational structure of both classes are similar, resulting in a possible mapping between both concepts.

Structural techniques are often built upon other mapping strategies like terminological ones, since structure-based techniques require previously deduced mappings to derive additional relations between entities which is exemplified in Figure 3.10, the initial mappings between the properties in this example might have been achieved by using string similarity measures.

3.3.3 Extensional Techniques

The approaches within this category use knowledge about the sets of individuals that belong to concepts and relations for aligning ontologies. Individuals or extensional information depend on the conceptual part of the ontology and can be used to precisely match classes since they are less prone to variations of respective ontologies [31].

If the to be matched ontologies have similar extension information, i.e. individuals which are used in both ontologies, the usage of instance-based matching techniques offers an easy way to find similarities between classes. An example of a possible mapping between two classes which share the exact same set of individuals and therefore can be considered to be similar, is illustrated in Figure 3.12.

But even if classes do not share a complete set of individuals with each other, instance-based matching techniques can be used to define an initial set of alignment anchors, which propose possible alignments to other mapping strategies.

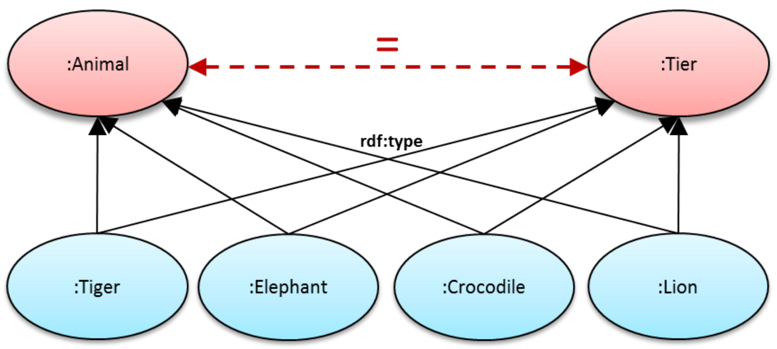


Figure 3.12: Two classes sharing the exact same set of individuals are considered to be similar

According to the matching technique classification shown in Figure 3.2, extensional techniques can be divided into:

Data analysis and statistical techniques: These techniques require a sample of a population (individuals) in order to be able to find regularities and discrepancies between them [31]. If such coherences were found, respective distances between those datasets can be computed e.g. using the Jaccard distance.

Language-based techniques: Once again, by identifying the structure of individuals of classes and extracting the literals of specific properties and comparing them with language-based techniques, additional similarities between properties and classes can be derived.

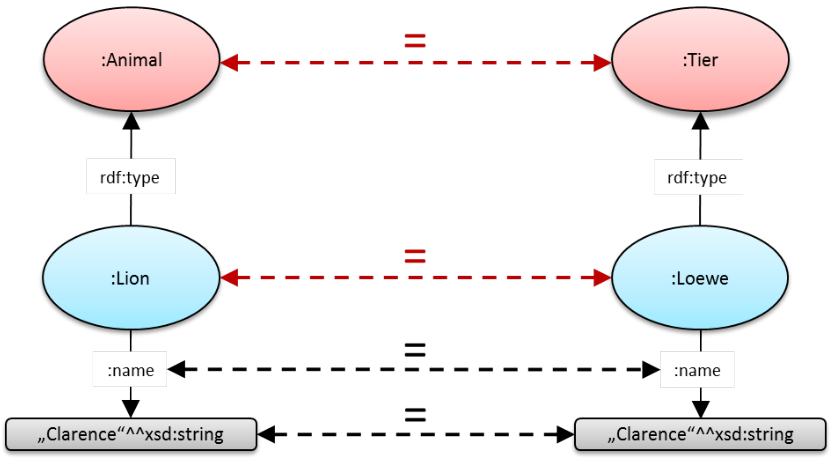


Figure 3.13: Using string similarity between elements of individuals to align classes.

Although instance-based matching techniques offer promising results, especially when used as support for other mapping strategies, they are often omitted as additional mapping strategy since extensional information is often!vote 1 not or just barely available and similarities between concepts detected by instance-based methods are only described by the degree of their relatedness to each other instead of a concrete relation definition [73]. In the present of a large set of extensional knowledge,

3.3.4 Semantic-based Techniques

The use of semantic-based matching techniques distinguishes ontology mapping from conceptual schema mapping, since semantic methods take advantage of the model theoretic semantics encoded in the ontologies. These semantics provide rules for interpreting the ontologies and furthermore provide the possibility to derive additional knowledge (relations, rules) between concepts in the ontologies [31].

These techniques can either use domain specific data like upper level or domain specific ontologies (WordNet [59, 67], SUMO [64], FMA [71]) or model-based techniques like reasoners (Pellet [66], FaCT++ [80], HermiT [74], ELK [51], TrOWL [79]) for refining alignment results or deducing new relations between entities of the ontologies.

We leave the domain of animals for a moment to illustrate the importance of semantic-based matching techniques for enhancing alignment results, which is depicted in Figure 3.14.

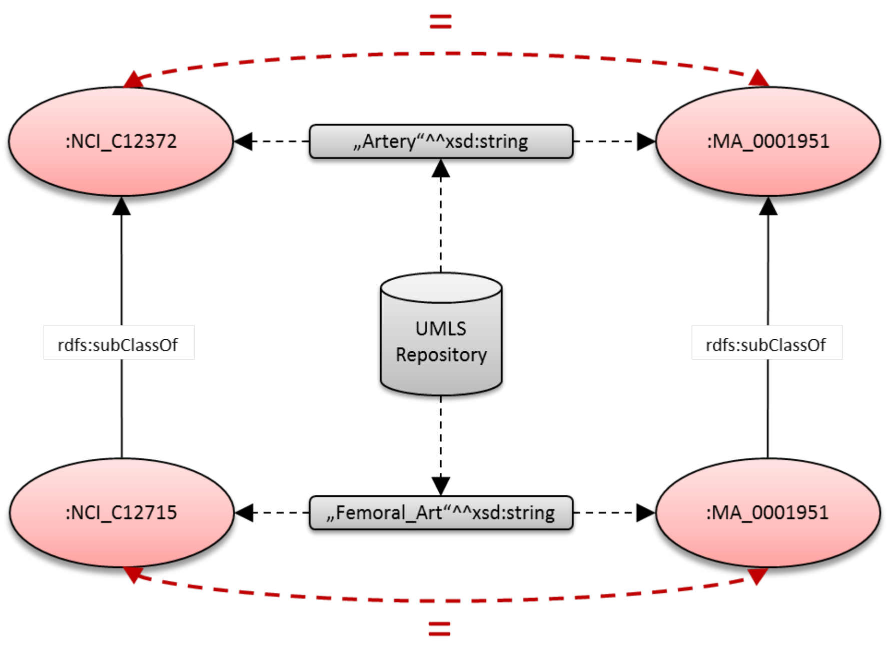


Figure 3.14: Finding mappings using domain specific background knowledge.

In this example, four classes from anatomy related ontologies, one describing the human anatomy and one describing the mouse anatomy, should be matched. Unfortunately in both ontologies there only exist class definitions and there obviously exists no string similarity between their entity names. In this particular case some matching techniques, e.g. terminological ones, would not be able to find any mappings between those classes.

But since semantic-based matching techniques allow the usage of external knowledge bases for supporting the mapping process, a domain related repository like the unified medical language system (UMLS) [7] can be used to detect similarities between those classes. A matching technique could exemplarily query that repository for the available entity names and check whether they describe the same anatomy part or not.

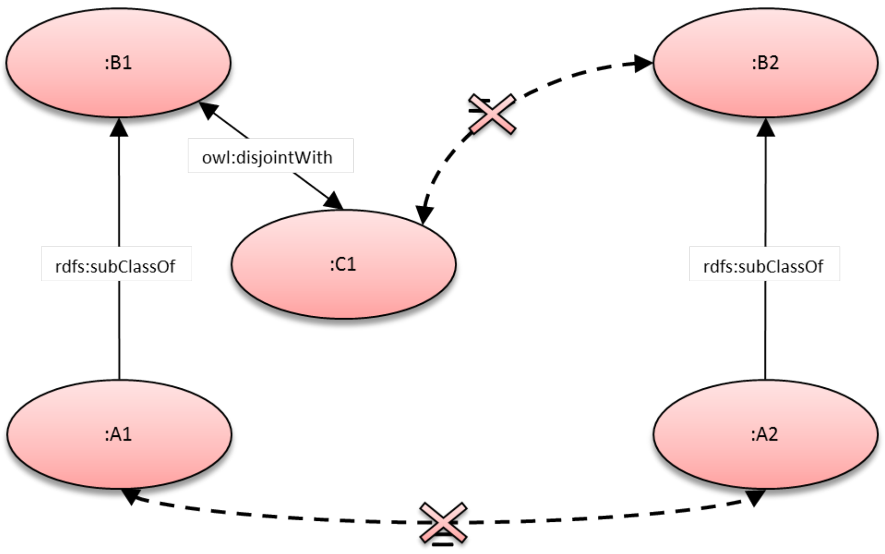


Figure 3.15: Finding inconsistencies using reasoning techniques.

Another example of using semantic-based techniques for refining alignment results is shown in Figure 3.15. An alignment between the classes `A1` and `A2` as well as one between `C1` and `B2` were proposed by a matching technique. Unfortunately both similarities cannot coexist since `A1` is a subclass of `B1` and `A2` is a subclass of `B2`, together with the disjointness between `B1` and `C1` it is not possible, that both similarities are valid at the same time without producing inconsistency within the ontologies. Such inconsistencies can easily detected using OWL reasoners.

Furthermore it is possible to use OWL reasoners for detecting instance alignments between two ontologies. Considering the triples shown in the two tables underneath and knowing that `:supervisedBy` is considered as `owl:FunctionalProperty` and is the `owl:inverseOf :supervises`, together with the instance alignment `:s1 owl:sameAs :s2` an OWL reasoner could infer that the triple `:p1 owl:sameAs :p2` holds, too.

Triples in O1

subject	predicate	object
:p1	rdf:type	:Professor
:s1	rdf:type	:Student
:s1	:supervisedBy	:p1

Triples in O2

subject	predicate	object
:p2	:supervises	:s2

Approaches for distributed or „loosely-coupled“ reasoning of ontologies along with mappings are mentioned in [84]. Here, mappings are described in the form of e.g. so-called bridge rules, rather than as native OWL axioms.

CHAPTER 4

Selected Ontology Matchers

In the present chapter we will briefly discuss several existing off-the-shelf ontology matchers, which we will subsequently combine. The six used ontology matchers are listed in Table 4.1 together with their implemented matching techniques.

We have chosen those six matchers primarily to ensure a heterogeneous set of ontology matchers but also for the sake of simplicity in terms of integrating them into one single ontology matcher, since many off-the-shelf matchers use the same internal Java libraries but often in different versions. Loading the same library several times, each time in another version into the build path of our project was unfortunately not realizable.

Ontology Matcher	terminological based	structure based	semantic based	instance based
Anchor-Flood	✓	✓	✗	✗
Aroma	✓	✓	✓	✗
Eff2Match	✓	✓	✓	✗
Hertuda	✓	✗	✗	✗
HotMatch	✓	✓	✓	✗
LogMap2	✓	✓	✓	✓

Table 4.1: List of used off-the-shelf ontology matchers and their used techniques

4.1 Anchor-Flood

Ontology Matcher	terminological based	structure based	semantic based	instance based
Anchor-Flood	✓	✓	✗	✗

Table 4.2: Matching techniques implemented in Anchor-Flood

4.1.1 Matcher Description

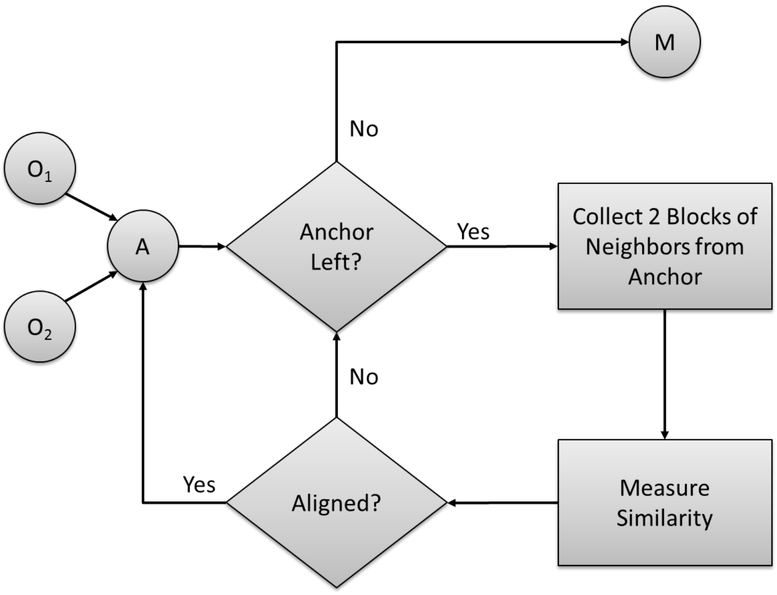


Figure 4.1: Anchor-Floods internal mapping process [44]

The mapping process of *Anchor-Flood* [44] as depicted in Figure 4.1, starts with an alignment (also called anchor) taken from an alignment set A and inspect its neighborhood (sub-/superconcepts, siblings and properties) for possible additional alignments. Since the size of such neighborhood adversely affects the runtime of Anchor-Flood, it is regulated by taking semantic similarity, retrieved by semantic matching techniques in terms of intrinsic information content, into account. The local alignment process itself is based on structural [**?**,8,38] and terminological [32,78,82] matching techniques.

If additional alignments were found, they get included into anchor set A and serve as future anchors for further processing. If no further anchors were left, the mapping process stops and returns the final set of alignments.

Although *Anchor-Floods* last participation in the OAEI campaign was in 2009, we used it for our matching approach due to its fast runtime and overall mapping performance[1].

4.2 AROMA

Ontology Matcher	terminological based	structure based	semantic based	instance based
Aroma	✓	✓	✓	✗

Table 4.3: Matching techniques implemented in AROMA

4.2.1 Matcher Description

In contrast to the majority of current state-of-the-art ontology matchers, *AROMA* [18] tries to find subsumption relations in addition to equivalence between entities (classes or properties) of two ontologies. Its approach relies on the assumption that:

An entity `A` *will be more specific than or equivalent to an entity* `B` *if the vocabulary (i.e. terms and also data) used to describe* `A`*, its descendants, and its instances tends to be included in that of* `B`*.* [18]

AROMAs Mapping process is divided into two main steps:

Term or datasets extraction. In this step, AROMA creates sets of relevant terms and data values for each entity in the to be matched ontologies by extracting the vocabulary of entities from their labels, individual values and comments.

Discovery of subsumption relations. In the second step, AROMA tries to discover subsumption relations by evaluating association rules between their respective relevant term- or datasets.

[1]http://oaei.ontologymapping.org/2009/results/

4.3 Eff2Match

Ontology Matcher	terminological based	structure based	semantic based	instance based
Eff2Match	✓	✓	✓	✗

Table 4.4: Matching techniques implemented in Eff2Match

4.3.1 Matcher Description

Eff2Match [14] (**Eff**ective and **Eff**icient ontology matcher) was developed by the Nanyang Technological University[2] and uses dynamic candidate reduction techniques for increasing efficiency as well as providing a high scalability. *Eff2Match* can be divided into four main stages which are depicted in Figure 4.2 and explained in the following:

Anchor Generation. In this stage, initial anchors are generated by using a string mapping technique. Names and labels of entities in the target ontology get normalized by removing delimiters and lower case conversion and afterwards populated into a hash table to be able to easily map them to their corresponding entities.

Candidate Generation. All entities which have not been matched in the previous stage are now enumerated using a Vector Space Model (VSM) [72] approach. More precisely, for each concept, VSM vectors for names, labels and comments get created in the concept itself and its ancestors and descendants. For the properties, these vectors get created in the property itself, as well as in its domain and range concepts.

Anchor Expansion. In this stage, additional entity mappings are detected by comparing source entities with their previous generated candidate entities by using terminological-based matching techniques.

Iterative Boosting. In the last step of the mapping process, an iterative boosting process is used to detect additional mappings using the anchor set A_{exp} by iteratively mapping concepts which have not already been matched with their respective candidates.

[2]http://ntu.edu.sg

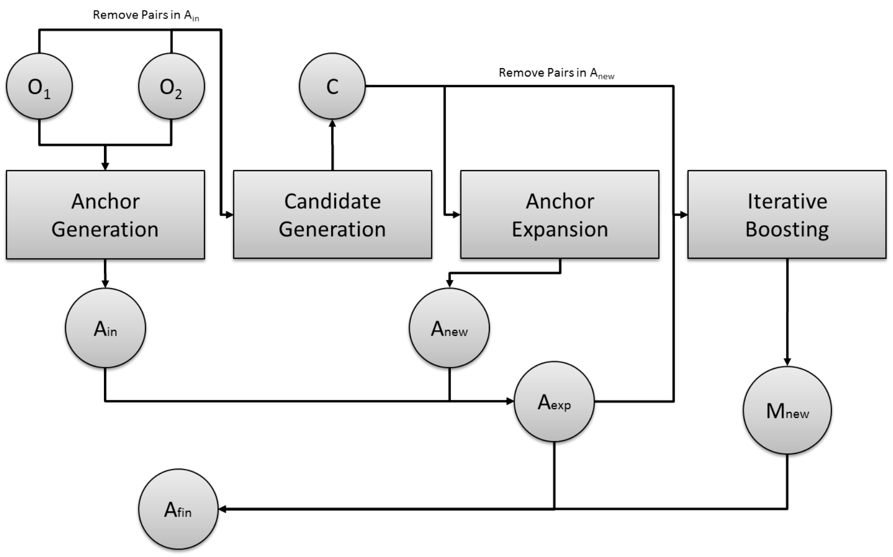


Figure 4.2: Eff2Matchs algorithm flow [14]

4.4 Hertuda

Ontology Matcher	terminological based	structure based	semantic based	instance based
Hertuda	✓	✗	✗	✗

Table 4.5: Matching techniques implemented in Hertuda

4.4.1 Matcher Description

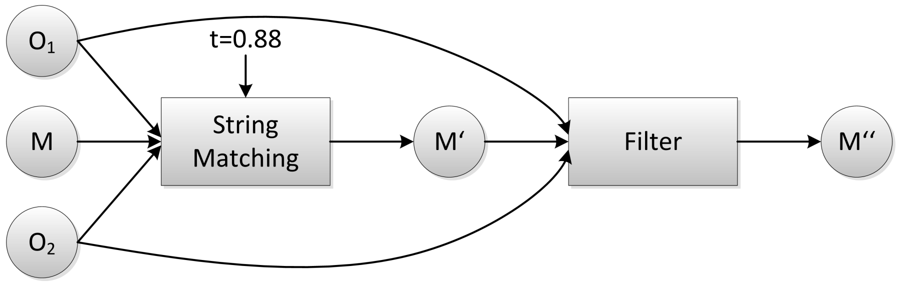


Figure 4.3: Hertudas mapping algorithm composition [46]

Hertuda [46] was developed by the Knowledge Engineering Group of Darmstadt University[3] and relies on a simple element based matcher with string comparison as shown in Figure 4.3. It only computes homogeneous mappings, i.e. class to class, datatype property to datatype property and object property to object property.

It iterates over all entities in the to be matched ontologies and compares them with each other using string comparison. This is done by extracting the URI, label and comment of each entity, comparing them with the respective opposite of other entities, calculating their similarities and only accepting an alignment if it exceeds a previous defined confidence threshold.

Before comparing the different elements of entities, *Hertuda* performs a preprocessing step. It tokenizes all elements, i.e. splits all terms with underscores, hyphens or which are written in camel case into two tokens and converts them into lower case. Therefore `has_Name`, `has name`, `has-Name` and `hasName` all result into the same tokens, `fhasg` and `fnameg`. For calculating the similarities a similarity matrix is computed. Unfortunately this approach of excessively iterating over all entities, leads to an enormous runtime increase and internal memory load if large ontologies were matched.

[3]http://www.ke.tu-darmstadt.de/

4.5 HotMatch

Ontology Matcher	terminological based	structure based	semantic based	instance based
HotMatch	✓	✓	✓	✗

Table 4.6: Matching techniques implemented in HotMatch

4.5.1 Matcher Description

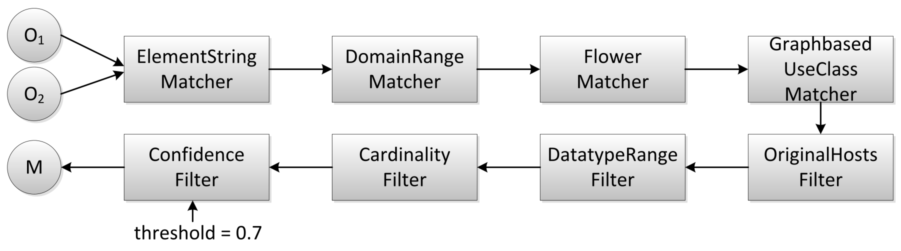


Figure 4.4: Hotmatchs mapping algorithm composition [17]

Like Hertuda, *Hotmatch* [17] was developed by the Knowledge Engineering Group of Darmstadt University. It basically consists of several mapping algorithms, which are executed sequentially during a mapping task and followed by some filters which aim to refine the mapping results and reduce the number of false-positive mappings.

In the following we will briefly describe the different mapping algorithms and filters of *Hotmatch* (cf. Figure 4.4).

ElementStringMatcher is exactly the same mapping algorithm as implemented in Hertuda. A string-based, element-level matcher which iterates over entities of ontologies and computes similarities of their URIs, labels and comments.

DomainRange Matcher is similar to the previous graph-based matcher, but instead of using class mappings as input, it uses property alignments. For every property mapping found by previous mapping algorithms, the property's domain and range are also mapped, inheriting the property mapping's confidence value.

FlowerMatcher is combining the previous mentioned approaches and extends them to the neighborhood of classes, i.e. their sub-/superclasses and properties, to determine similarity between classes.

GraphbasedUseClassMatcher is a graph-based matcher and tries to find alignments between properties. Therefore it takes class mappings found by previous algorithms as input and iterates over all properties, checking whether their domains and ranges are equals or not. If so, the confidence of the new mapping between the two properties is the mean value between the confidence of the respective domain and range mappings.

OriginalHostsFilter ensures that only mappings which directly align entities of the two ontologies were accepted for the final alignment set.

DatatypeRangeFilter removes datatype property mappings, where the matched datatype properties have different datatypes.

CardinalityFilter enforces 1-to-1 alignments by removing multiple mappings of the same entity, only keeping the alignment with the highest confidence value.

ConfidenceFilter discards all alignments which are not able to exceed a specific confidence value threshold.

4.6 LogMap2

Ontology Matcher	terminological based	structure based	semantic based	instance based
LogMap2	✓	✓	✓	✓

Table 4.7: Matching techniques implemented in LogMap2

4.6.1 Matcher Description

We now provide an overview of the main steps performed by *LogMap2* [49], which are schematically represented in Figure 4.5

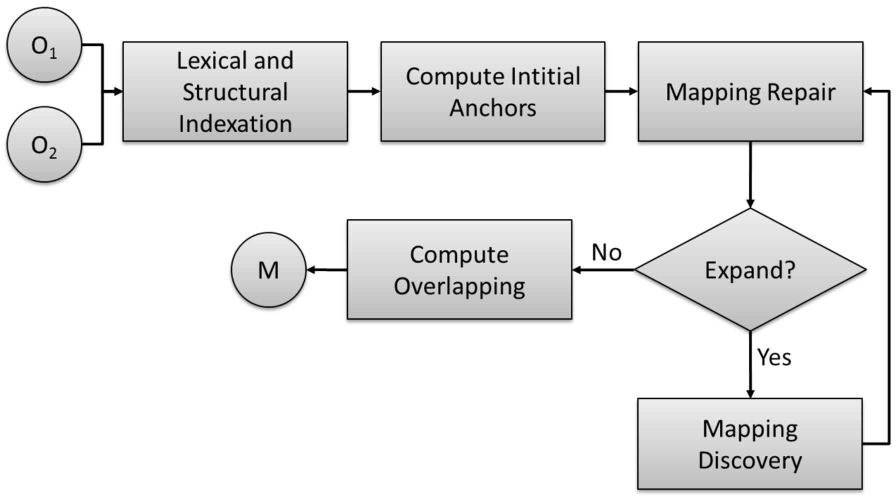


Figure 4.5: LogMap2s mapping workflow [49]

Lexical indexation. LogMap2 tries to index labels, as well as their lexical variations, of classes in both ontologies and therefore allows the enrichment of those indices by using a external lexicon like WordNet.

Structural indexation. An interval labeling schema [1, 13, 61] is used by LogMap2 to represent the extended class hierarchy of both input ontologies. The extended hierarchies are computed by either using structural heuristics or a DL reasoner.

Computation of initial anchor mappings. By intersecting the lexical indices of both input ontologies, LogMap2 computes an initial set of anchor mappings, which serve as start for additional alignments.

Mapping repair and discovery. This step represents the major part of LogMap2's mapping process and consists of an iterative process which combines repair and discovery algorithms. For the repair part, LogMap2 uses a reasoning algorithm to detect unsatisfiable classes regarding the two input ontologies and mappings found so far and tries to repair them by using a greedy diagnosis algorithm [49]. In the discovery step, LogMap2 relies on the *principle of locality*, i.e. if there exists an alignment between two classes `C1` and `C2`, then the classes semantically related to `C1` and those of `C2` are very likely to be mapped too.

If no further knowledge was created in the discovery part, the process stops and returns a set of alignments which are very likely to be clean, i.e. do not produce unsatisfiable assumptions.

CHAPTER 5

Mix'n'Match: An Approach for Combining Ontology Matchers

In this chapter, we present our combined approach to ontology mapping, which we call *Mix'n'Match*. Our goal is an approach that combines existing off-the-shelf ontology matchers with all their strengths and weaknesses in a unified manner. Instead of finding the one "best off-the-shelf matcher for the problem at hand" we aim at combining matcher results of different matchers.

Intuitively, the idea of Mix'n'Match is very simple: starting from an empty set of alignments, we aim at iteratively supporting in each round, matchers with the combined results of other matchers found in previous rounds, somehow aggregating the results of a heterogeneous set of ontology matchers.

A schematic overview of the overall Mix'n'Match framework is shown in Figure 5.1.

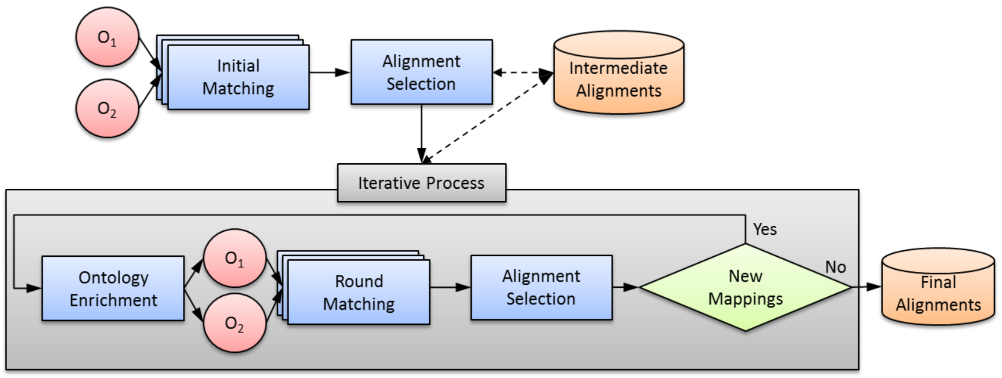


Figure 5.1: Framework of Mix'n'Match

In the following we will describe each component of our framework in more detail, discuss alternative solution approaches and explain why we have decided to implement specific strategies. In the end of this chapter we will discuss the advantages of an anytime behavior for our matching approach and illustrate its feasibility based on a motivating example.

5.1 Initial Matching & Round Matching

The core idea of our matching approach is the iterative execution of several off-the-shelf ontology matchers and the aggregation of their mapping results. Since *Mix'n'Match* is not restricted to a specific amount of ontology matchers, it can basically be extended by an arbitrary amount of individual ontology matchers in order to produce even more accurate alignment results. The downside of this feature is the increasing runtime which is necessary in order to complete the mapping task, the more matchers are used.

In this section we will describe three different matcher execution approaches and discuss their advantages and disadvantages in more detail.

5.1.1 Running Ontology Matchers in a Sequential Way

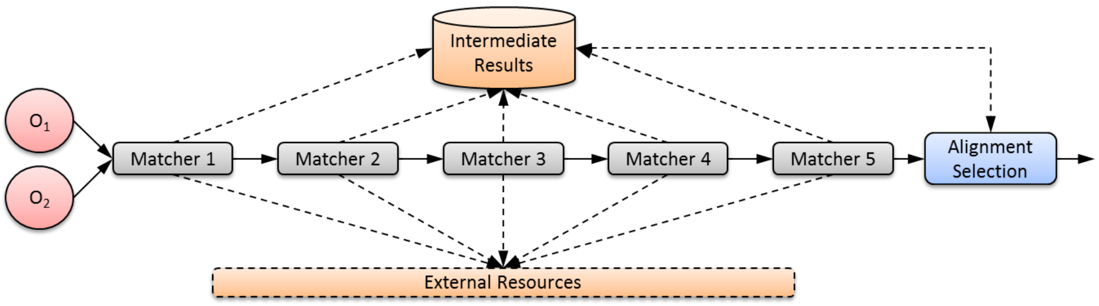


Figure 5.2: Running the individual ontology matchers sequentially.

Our first approach, which we have implemented, was the sequential execution of the ontology matchers and is illustrated in Figure 5.2. In this simple execution strategy an ontology matcher starts its mapping process if the previous one has finished its mapping task.

Advantages: It is a very simple and easy to implement execution approach and ontology matchers which need to access external resources like *WordNet* do not interfere each other.[1]

Disadvantages: It is the most time consuming execution approach discussed in this thesis, since the overall runtime is basically an aggregate of every individual ontology matcher runtime. Furthermore the single off-the-shelf ontology matchers are independent in their mapping task so they would not have to wait on each others results, making a sequential execution of matchers even more unfeasible.

5.1.2 Parallel Execution of Ontology Matchers

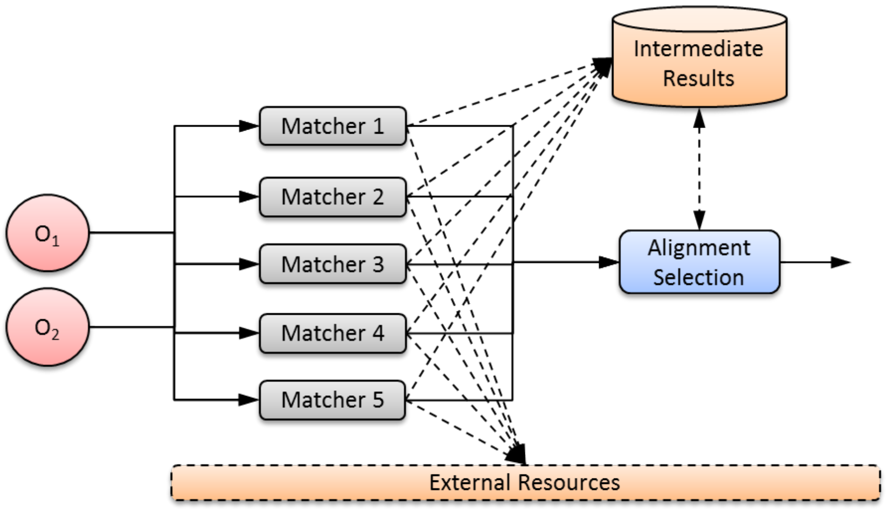


Figure 5.3: Using a parallel execution approach.

The current implemented ontology matcher execution approach within *Mix'n'Match* is a parallel one as depicted in Figure 5.3. In this execution strategy the individual matchers are started simultaneously using multi-threading.

Advantages: In contrast to the sequential execution approach, this strategy needs way less time to complete its overall mapping task, primarily because the single ontology matchers won't have to wait on each others completion to start their mapping process.

Disadvantages: It is not as easy to implement like the previous approach and heavily relies on the characteristics of the evaluation machine (the more cores the more parallel threads). Another drawback of this approach, which we have discovered during our evaluations, is the interference among individual matchers

[1]Interfering in terms of overwriting temporary changes performed in the external resource

when using external resources. If two or more matchers use the same external resource (e.g. *WordNet*) they sometimes crash due the simultaneously access on that particular external resource based on our implementation strategy.

Although we had to remove some matchers from our matching approach, caused by this type of errors, we have chosen this execution approach over the sequential one due its significant runtime decrease.

5.1.3 Performing a Distributed Execution of Ontology Matchers

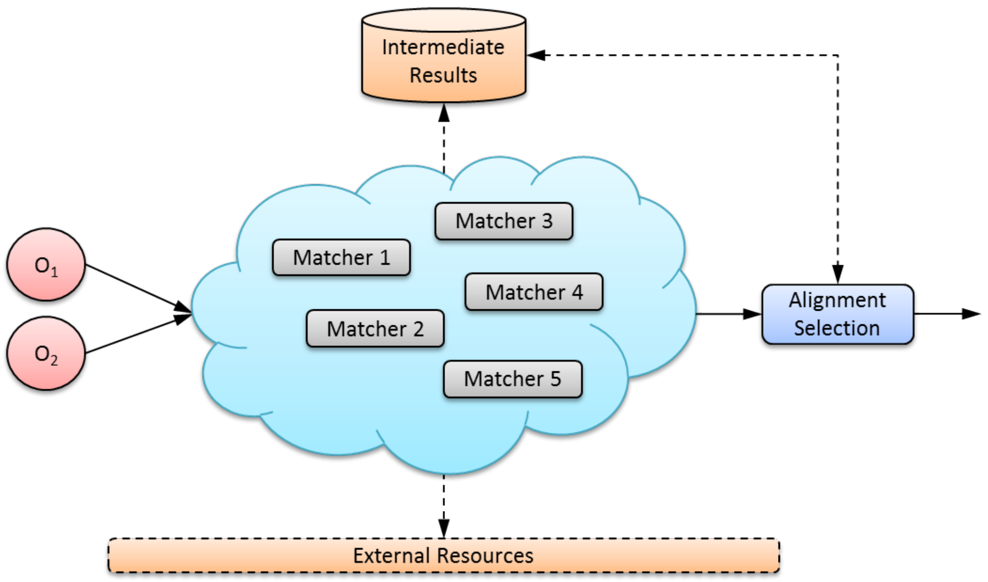


Figure 5.4: Transferring the matchers into the cloud and running them in a distributed manner.

This last execution strategy is probably the most promising one and could finally solve the runtime problems of *Mix'n'Match*, because even with the multi-threading approach proposed above we were not able to decrease the runtime of our approach to that of the slowest used off-the-shelf ontology matcher (cf. Figure 5.4). In future versions of *Mix'n'Match*, we will evaluate this strategy in more detail.

Advantages: As mentioned above, the hypothesis is that we would be able to decrease the runtime of *Mix'n'Match* to that of the slowest used off-the-shelf matcher by using a distributed execution approach. Furthermore we could solve the interference issues mentioned above as well as some build-path library dependency issues, which are caused by the usage of different versions of the same library by some ontology matchers since every matcher would run in its own environment not being able to interfere with any others resources but its own.

Disadvantages: A possible disadvantage of this approach is the exploration of a suitable distribution solution. Strategies like Map&Reduce [19], which we thought of in an early brainstorming phase simply does not fit our needs.

5.2 Alignment Selection

In each round we need to decide which alignments to keep and which ones to ignore from the union of new alignments found by all considered matchers. This decision is very crucial since we encode accepted alignments into the ontologies as additional knowledge and rerun the whole mapping process again. If we could not ensure a high precision of the accepted alignments, we would encode wrong knowledge into the ontologies, which would probably lead to even more wrong new alignments.

In this section we will describe four different alignment selection methods and discuss their strengths, weaknesses and our choice in more detail.

5.2.1 Performing a Majority Vote for Selecting Alignments

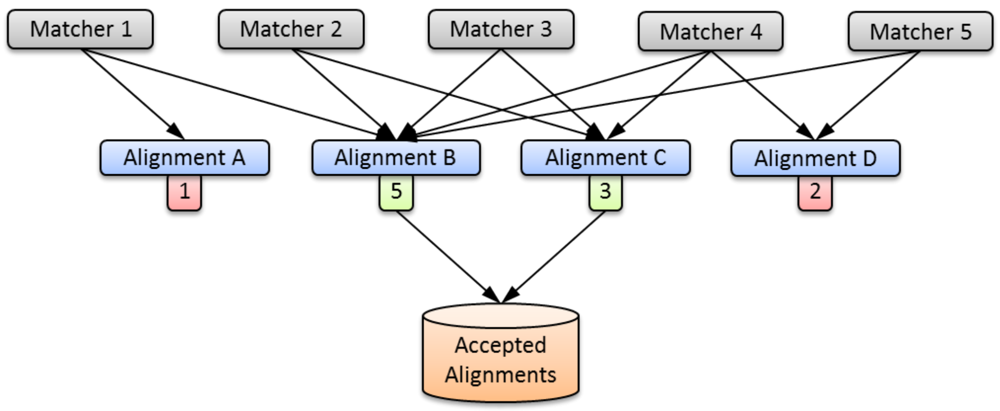


Figure 5.5: Accepting only those alignments, which were found by the majority of the ontology matchers.

An interestingly easy but effective selection approach is a majority vote (cf. Figure 5.5). Using such a voting strategy for alignment selection means, that only those alignments were accepted which were found by a specific number (i.e. the majority) of the used off-the-shelf ontology matchers. Other thresholds rather than using the majority of ontology matchers, i.e., accepting mappings confirmed by a certain number n of matchers are also feasible and have to be evaluated for the specific mapping task at hand. We have used specifically $n = m/2$ (where m is the amount of matchers in the matcher portfolio) and $n = 1$ (i.e. accept all mappings found by any matcher), for further Details see Section 6.2 below. This strategy is currently responsible for selecting alignments in *Mix'n'Match*.

Advantages: The probably biggest advantage of this approach, besides its simplicity, is the high precision of the accepted alignments. The more heterogeneous matchers are used within the mapping process, the smaller are the odds that alignments found by the majority of those matchers are wrong. Additionally this approach can be executed in a complete unsupervised way, offering the possibility to run a completely automatic mapping process. Furthermore the results of single off-the-shelf matchers which produce very bad results on specific types of datasets are not taken into account, but . . .

Disadvantages: . . . this holds for the results of single outstanding ontology matchers, which find alignments no other matcher was able to detect, too. Additionally the high precision of the found alignments mentioned above, usually comes in hand with a small recall of those mappings which we partially are able to increase with our iterative execution approach.

5.2.2 Using a Weighted Majority Vote Approach

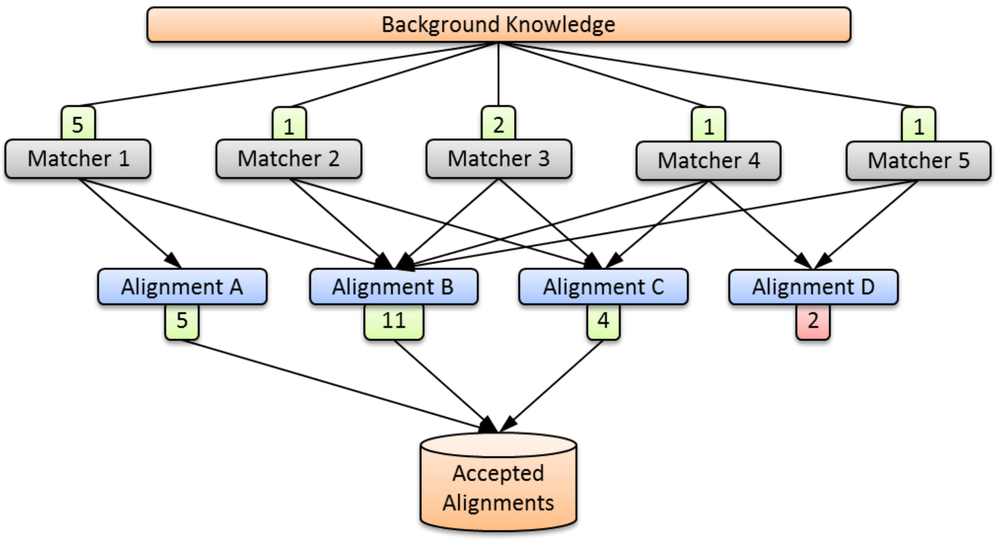


Figure 5.6: Weighting ontology matchers based on their individual performance.

To tackle one of the major disadvantages of the simple majority vote approach mentioned above, a weighted majority vote strategy could be used(cf. Figure 5.6). Using this strategy, single matchers get weighted based on background knowledge (i.e. their performance on specific mapping tasks) and therefore have different impact on the mapping scenario at hand. Possible strategies of such weighting approaches are proposed in [43] and [15].

Advantages: Like the normal majority vote, this adapted version could produce a very highly precise alignment set and additionally could be able to take the results of single outstanding ontology matchers into account.

Disadvantages: Nevertheless such a weighting of ontology matchers based on the characteristics of the to be matched datasets requires some background knowledge, which is usually not able to be gathered in a fully automatic and unsupervised way. In order to preserve these abilities the authors of [43] and [15] have proposed some solution strategies which have to be evaluated and tested in future versions of *Mix'n'Match*.

5.2.3 Train a Machine Learning Classifier for Alignment Selection

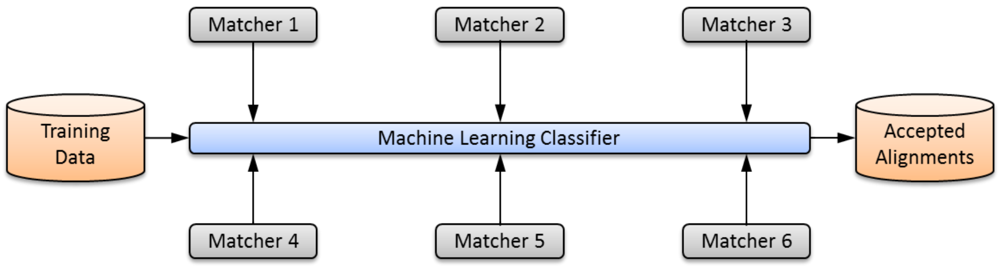


Figure 5.7: Training a machine learning classifier which selects alignments.

Another promising strategy for selecting alignments is using machine learning, as proposed in [25, 27, 28] and exemplified in Figure 5.7. Where the idea is to train a classifier, which then has to decide whether or not an alignment of a specific matcher gets accepted or not.

Advantages: In difference to the weighted majority vote approach, it could be possible to train some sort of universal classifier which is able to classify alignments of any kind of datasets. This could lead to better results than the previous approaches, if suitable training data is available.

Disadvantages: Of course the necessity of a sufficiently dimensioned set of training data makes it difficult to run this selection approach in a completely automatic way. Furthermore the selection of an accurate feature set is crucial and difficult since it has to fit the characteristics of the to be matched ontologies as shown in [27], where other features than just the confidence value like in [26, 56], were chosen and produced way better results.

5.2.4 Involving Users to Support the Selection Process

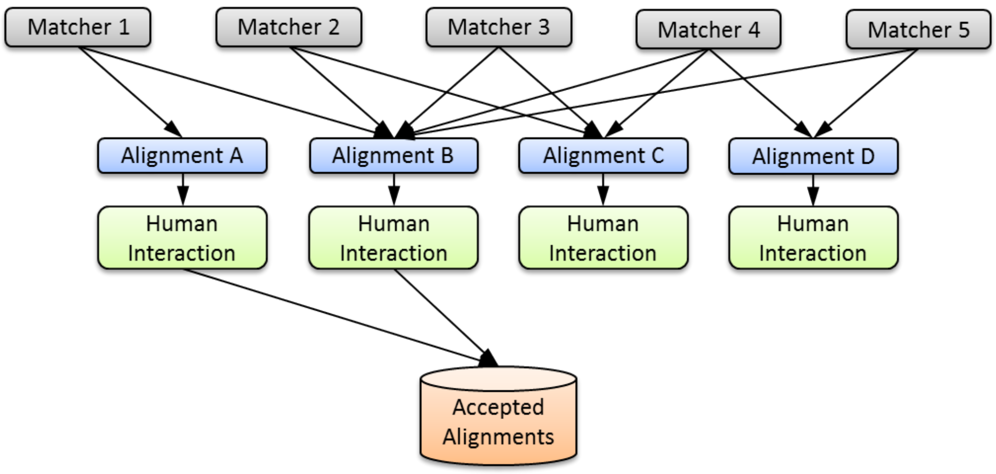


Figure 5.8: Involving domain experts into the selection process.

This strategy is probably the most accurate and precise one among all discussed approaches (cf. Figure 5.8). The main working principle is simple, it collects all alignments produced by the individual ontology matchers and then let a human domain expert decide whether or not an alignment gets accepted.

Some research was dedicated to involving users into the mapping process but mostly focused on design-time matcher interaction [23,65].

Advantages: As mentioned above, on the one hand this approach - under the assumption that a human domain expert knows exactly which alignments are true - would produce the alignment set with the highest precision, . . .

Disadvantages: . . . but on the other hand is unfeasible for large ontology tasks, where an user would have to check thousands of alignments. Additionally the implementation and representation of such an user interaction module is quite difficult and requires a lot of experience in user interaction design.

Generally speaking, this approach might be an effective extension to other selection approaches, where alignments which barely won't be accepted get passed to the user for further processing as already implemented in [49], but should not be used as only selection approach where alignments get passed to without being filtered.

5.3 Enriching Ontologies with Additional Knowledge

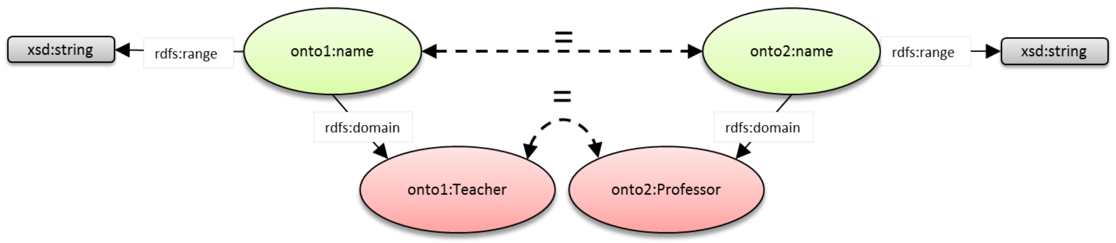


Figure 5.9: Mappings used to exemplify the enrichment strategies.

In order to be able to use additional knowledge in terms of input alignments for our mapping process, all used ontology matchers would have to support the import of such input alignments, which was unfortunately only supported by Anchor-Flood [44]. Therefore we needed to find another way to "inject" alignments in a non-obtrusive way[2] into the mapping process.

5.3.1 Native OWL Axioms

Adding support for reference alignments (without touching the matchers' source code) was not straightforward. Our first idea was to add reference alignments explicitly in the form of OWL equivalences into the ontologies, using properties `owl:equivalentClass` (for mapping concepts), `owl:equivalentProperty` (for mapping properties), or `owl:sameAs` (for mapping individuals).

An example implementation of the mappings depicted in Figure 5.9 is shown in Listing 1 and respectively Figure 5.10 where the two equivalence properties are already included into *O1*.

[2]In order to keep our framework general and extensible, we do not want to modify the code of the ontology matchers or interfere in some other tool-specific way with the matchers used in our framework.

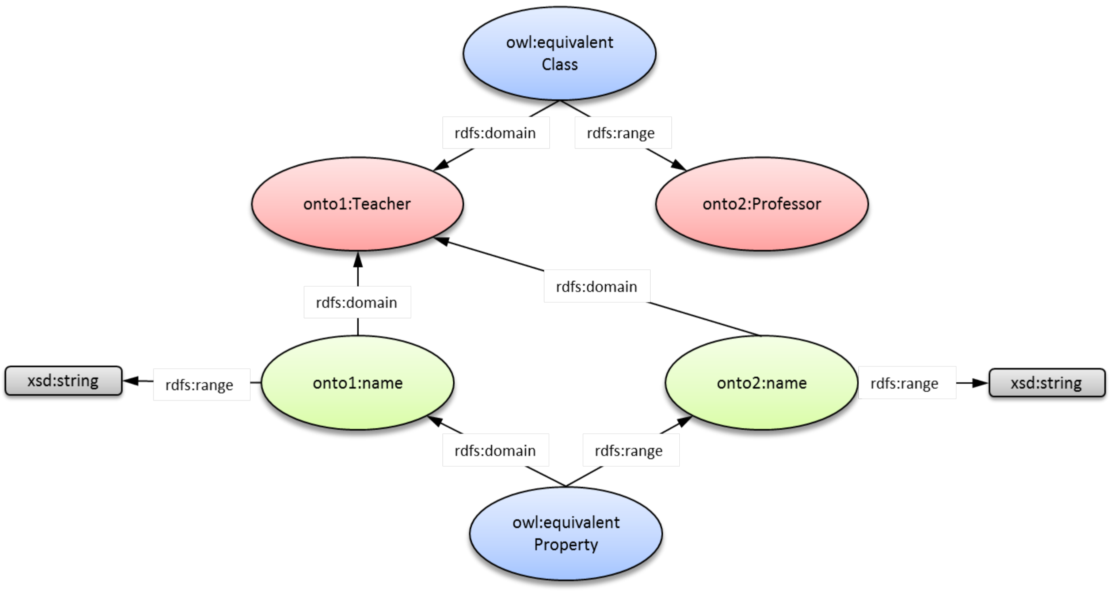


Figure 5.10: Stating the equality between entities via OWL axioms in one ontology.

Example 1. Implementing this strategy would mean to simply add to both ontologies *O1* and *O2* e.g. the following OWL statement after the first mapping round (and hoping it would be considered for subsequent reruns of the considered matchers):

```
...
@prefix : <http://ontology.org/onto1/> .
@prefix onto2: <http://ontology.org/onto2/> .
...
:Teacher a owl:Class;
    owl:equivalentClass onto2:Professor.

:name a owl:DatatypeProperty;
    rdfs:domain :Teacher; rdfs:range xsd:string.
    owl:equivalentProperty onto2:name.

onto2:name a owl:DatatypeProperty.

onto2:Professor a owl:Class.
...
```

Advantages: Using OWL axioms for stating the equality between entities is the most simple and obvious approach and furthermore supports the definition of one-to-many and many-to-many alignments as well as complex mappings.

Disadvantages: However, this approach to "emulating" reference alignments proved unsatisfactory, since we would be referring in *O1* to entities from *O2* and vice versa.

Firstly, such referring would need additional definitions (`owl:Class`, `owl:DatatypeProperty`, or `owl:ObjectProperty` of the referred entities in the respective other ontology of all aligned entities, since otherwise the such enriched ontology would no longer satisfy the syntactic restrictions of OWL DL: obviously, this is inconvenient, because it would lead to significant overhead essentially redefining already existing entities.

Secondly, even if we would have redefined the equal entities, further mapping rounds would always align the duplicate entities as preliminary tests have shown.

5.3.2 URI Replacement

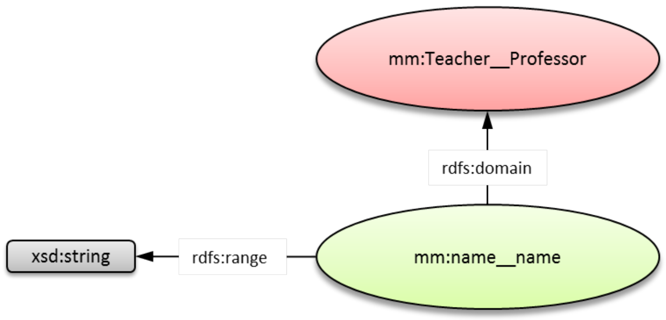


Figure 5.11: Replacing the URIs of equal entities by a normalized one.

Somewhat surprisingly, an alternative, much simpler approach to emulating reference alignments proved to be much more effective than adding OWL equivalence axioms: instead of "axiomatizing" alignments using OWL, we just created for each reference alignment a combined new URI for the matched entities, which was then replaced within both *O1* and *O2* as illustrated in Figure 5.11 and Listing 5.1.

Example 2. Based on a possible found alignment in the previous round we modify both ontologies by replacing the URIs of the two classes *Teacher* and *Professor* as well as the two properties *name* and *name*, assuming alignments between these entities were found, by unified ones. <*http://example.org/MixMatch/Teacher__Professor*> and <*http://example.org/MixMatch/name__name*>, cf. Listing 5.1

Listing 5.1: Ontology *O*1 after enrichment

```
...
@prefix : <http://ontology.org/onto1/> .
@prefix mm: <http://example.org/MixMatch/> .
...
mm:Teacher__Professor a owl:Class.

mm:name__name a owl:DatatypeProperty;
    rdfs:domain mm:Teacher_Professor; rdfs:range xsd:string.
...
```

Advantages: This approach is quite easy to realize and produces the smallest overhead of all proposed enrichment methods since it does not add additional axioms to the ontologies. Additionally, by replacing the URIs from matched entities by an unified one, it states them as equal in the sense of URIs as global identifiers in the World Wide Web.

Disadvantages: Because this strategy directly replaces the URIs of two matched entities it is restricted to one-to-one mappings and not able to detect one-to-many or many-to-many alignments. We will discuss a possible solution approach for this drawback in Chapter 8.

Nevertheless many matchers seem to ignore URIs as unique identifiers of entities, i.e. they still would consider the class *<http://ontology.org/onto1/Teacher>* mentioned in ontology *O1* different from the same class mentioned in *O2* in the input. In fact, only YAM++ seems to recognize URIs as really unique: we verified this experimentally by normalizing the URIs of completely unrelated entities; if the matchers produced alignments of those entities and omitted probably correct ones in favor of these incorrect alignments, then we considered this matcher as treating URIs as really unique.

But despite the above-mentioned fact that most matchers seem to ignore URIs as unique identifiers of entities, our experiments showed that URI replacement was effective in boosting the confidence in such asserted alignments in almost all considered matchers.[3]

5.3.3 Entity Annotations

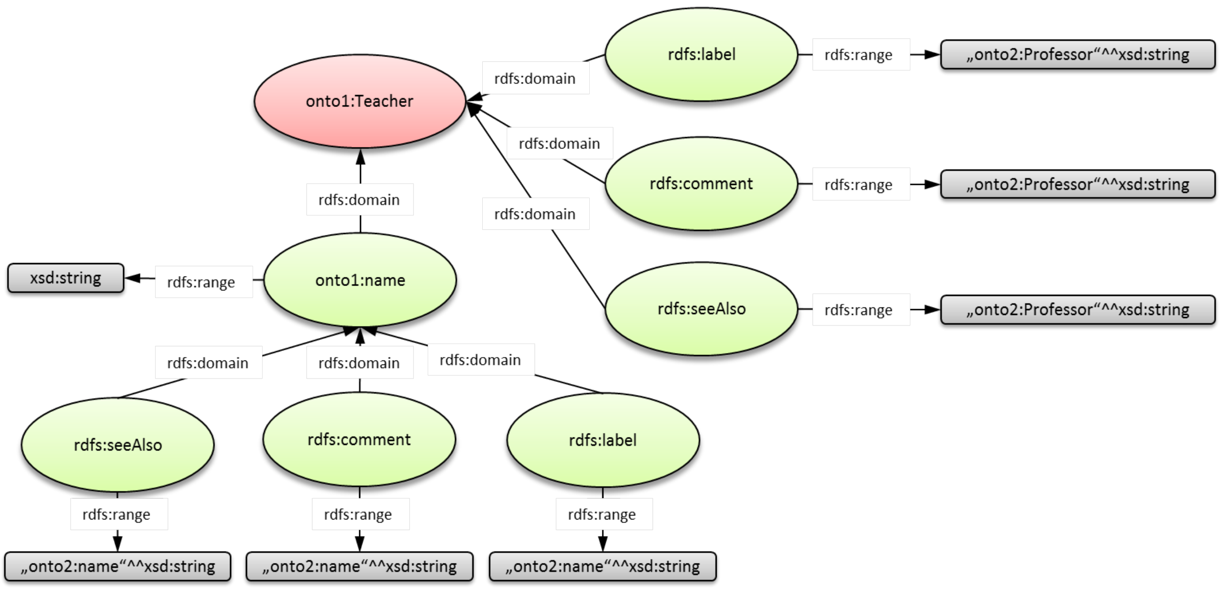


Figure 5.12: Adding annotations to matched entities, referring to their counterpart.

[3]Deliberately leaving out internal details of the respective matchers, we may only conjecture here that other features like string-mapping were triggered after reference alignments being "asserted" by URI replacement, which consequently lead to further new alignments being found in subsequent calls.

Using this strategy, we would simply add several annotations in the form of `rdfs:label`, `rdfs:comment` and `rdfs:seeAlso` tags to the matched entities and refer to their corresponding counterpart which is exemplified in Figure 5.12 and Listing 5.2

Example 3. Based on a possible found alignment in the previous round we modify both ontologies by adding annotations to all entities to whom alignments were found, referring to their mapping counterparts.

Listing 5.2: Ontology *O*1 after enrichment

```
...
@prefix : <http://ontology.org/onto1/> .
...
:Teacher a owl:Class;
    rdfs:label "onto2:Professor";
    rdfs:comment "onto2:Professor";
    rdfs:seeAlso "onto2:Professor".

:name a owl:DatatypeProperty;
    rdfs:domain :Teacher; rdfs:range xsd:string;
    rdfs:label "onto2:name";
    rdfs:comment "onto2:name";
    rdfs:seeAlso "onto2:name".
...
```

Advantages: Once again, this approach is quite easy to implement and understand and since it just uses annotations for referring to the matched counterparts there is no necessity for redefining matched entities like when using OWL equality axioms.

Disadvantages: This strategy of using annotations for encoding alignments into the ontologies heavily relies on the nature of the used ontology matchers. If those matchers do not take annotations into consideration for their mapping task, this approach is completely useless.

We assume that this strategy could primarily be used as addition to other enrichment strategies, since it is completely independent from any other approach but useless if the majority of matchers does not consider annotations in their mapping process.

5.4 Anytime Behavior in Mix'n'Match

Based on the nature of our matching approach of running several off-the-shelf matchers in an iterative way, *Mix'n'Matchs* runtime performance is quite bad in comparison to other state of the art ontology matchers, especially by mapping large ontologies (cf. Evaluation Chapter 6). Its performance becomes even worse, the more matchers are called and executed in its iterative process.

To overcome this issue or at least to reduce its impact on the mapping results, we implemented the possibility to interrupt the mapping process at any time and still being able to provide mapping results. This behavior is called *Anytime Behavior* and was initially used in the domain of search algorithms to offer them the capability to trade runtime for quality of results [20,83].

Although this strategy sounds quite simple, it had to be harmonized with our majority vote alignment selection approach, which we realized by storing all intermediate mapping results of all matchers and keeping track of the number of ontology matchers which have found these alignments over the last mapping rounds.

An example scenario which illustrates the feasibility of this approach is illustrated in Figure 5.13.

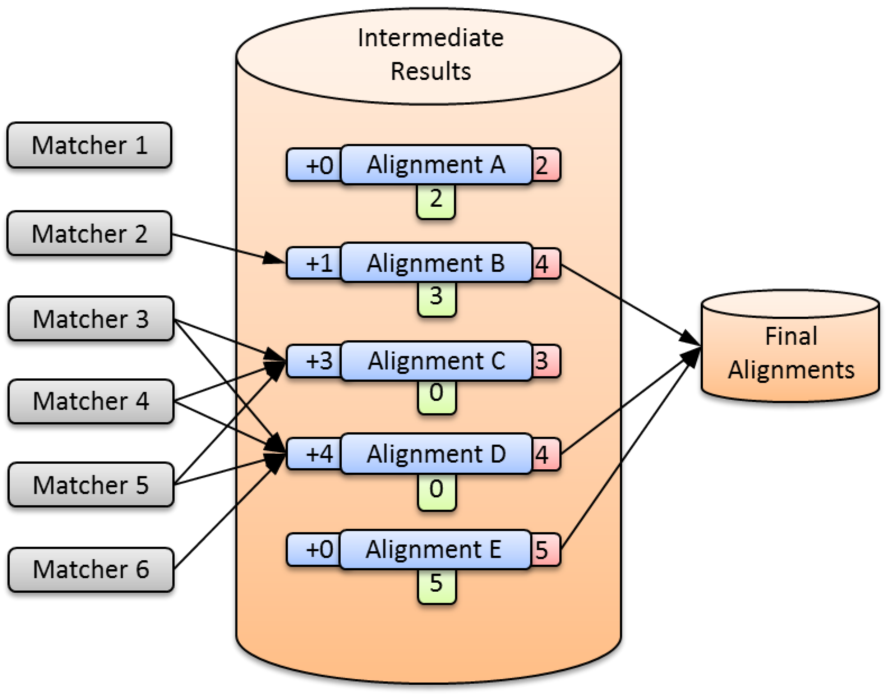


Figure 5.13: Exemplary illustration of anytime behavior in *Mix'n'Match*

Four out of six matchers had already finished their mapping tasks and added their votes to the intermediate mapping results[4] at the time the interruption took place, which results in the acceptance of alignments B,D and E since they now fulfill the necessary majority constraint. If we would not consider these intermediate round re-

[4]Of course they only voted for those alignments, which they have not already voted for.

sults, alignments B and D were not accepted for the final alignment set, although they would have passed the majority vote after the current mapping round had finished.

CHAPTER 6

Empirical Evaluation of Mix’n’Match

In the following chapter we will discuss results gathered by evaluating our approach on various OAEI evaluation tracks in comparison to selected ontology matchers and show that we were either able to outperform them in terms of F-measure or at least be competitive with the respective top-performing ontology matchers.

6.1 Preliminaries

6.1.1 Evaluation System

The system specifications of the machine that has been used to perform evaluations are listed in Table 6.1. Note that this machine has specifications similar to those of a common workstation and was not optimized for evaluation purpose.

System Specifications	
Processor	Core 2 Duo E6850
Processor Details	3.00 GHz (4 MB L2, 1333 MHz FSB)
RAM	4 GB
Operating System	Windows 7 64 Bit

Table 6.1: System specifications of the evaluation machine

6.1.2 Evaluation Approach

For our evaluations we ran each ontology matcher independently through Eclipse on the respective datasets and gathered their runtime as well as calculated the precision, recall and F-measure values of their produced alignments by using provided reference alignments.

In order to be able to understand these evaluation metrics, we briefly discuss them in the next section.

6.1.3 Precision, Recall and F-Measure

We have measured the performance of each ontology matcher by the precision, recall and F-measure value of its produced alignments. These metrics are common for information retrieval tasks and were firstly mentioned in [69] before they got adapted for ontology mapping tasks in [22].

Basically they are measuring the completeness and correctness of produced alignments by comparing them with a set of available reference alignments.

6.1.3.1 Precision

Precision measures the ratio of correctly found alignments (true positives) over the total number of produced alignments (true positives and false positives). An alignment set with a high precision will most likely does not contain all available alignments, leading to a lower recall.

Definition 7 (Precision)**.** *Given a reference alignment R, the precision of some alignment A is given by*

$$Pre(A, R) = \frac{|R \cap A|}{A}$$

6.1.3.2 Recall

Recall measures the ratio of correctly found alignments (true positives) over the total number of available alignments (true positives and false negatives). An alignment set with a high recall will most likely contain many false alignments, leading to a lower precision.

Definition 8 (Recall)**.** *Given a reference alignment R, the recall of some alignment A is given by*

$$Rec(A, R) = \frac{|R \cap A|}{A}$$

6.1.3.3 F-Measure

Unfortunately solely relying on recall and precision for comparing ontology matchers has its drawbacks. Maybe an ontology matcher which performs well in terms of precision has a very low recall and another one has a very low precision but high recall.

In order to be able to compare such systems by their precision and recall values and moreover be able to adjust parameters which affect the results in a way that both values are optimal, a combined metric of precision and recall called F-measure can be used.

Definition 9 (F-Measure)**.** *Given a reference alignment R, precision and recall, the F-measure of some alignment A is given by*

$$F(A,R) = \frac{2 \times Pre(A,R) \times Rec(A,R)}{Pre(A,R) + Rec(A,R)}$$

The more general definition of F-measure takes an additional parameter α into account which offers the possibility to weight precision and recall and therefore align their importance accordingly. For our evaluations we set α to 0.5, i.e. weighting precision and recall equally; in that case the F-measure value represents the harmonic mean of precision and recall.

6.2 Ontology Alignment Evaluation Initiative (OAEI)

Within this section, we will evaluate our approach on three different datasets provided by the OAEI and compare its results with those produced by the off-the-shelf ontology matchers used within Mix'n'Match.

6.2.1 Datasets

Dataset Characteristics				
Dataset	Formalism	Relations	Language	Number of Ontologies
Benchmark	OWL-DL	=	EN	5
Conference	OWL-DL	=, <=	EN	7
Anatomy	OWL	=	EN	2

Table 6.2: Characteristics of used datasets

The main characteristics of the three datasets used for evaluation are summarized in Table 6.2. The column „Formalism“ indicates the complexity of ontologies, „Relations“ lists the possible relations between entities in the ontologies, „Language“ states the language in which the ontology was written and „Number of Ontologies“ shown the number of ontologies within the dataset. These datasets consist themselves of several ontologies which have to be matched either against each other or against one base ontology. The characteristics of the ontologies within each dataset are described in the following.

6.2.2 Benchmark Dataset

The Benchmark dataset[1] was one of the first datasets used by the OAEI for evaluating ontology matchers and describes the domain of bibliographic references. The mapping relations are depicted in Figure 6.1 one base ontology (onto.rdf) is matched against four real-world bibliographic ontologies, namely BibTeX/MIT(onto301.rdf), BibTeX/UMBC(onto302.rdf), the Karlsruhe bibliographic ontology (onto303.rdf) and the INRIA bibliographic ontology (onto304.rdf). All five ontologies contain at most 166 entities (cf. Table 6.3), which is a quite small number compared to other benchmarks and therefore results in a relatively short mapping time.

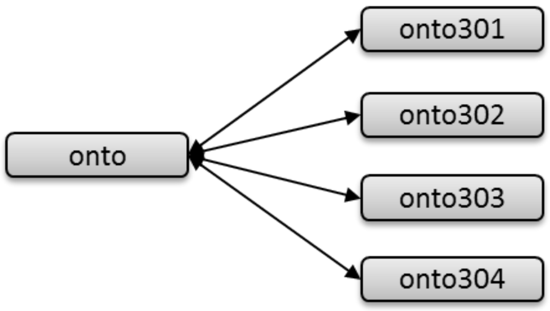


Figure 6.1: Mapping relations between the ontologies within the Benchmark track

Size of Benchmark Ontologies

Ontology Name	Number of Classes	Number of Properties	Number of Individuals	Number of Entities
onto.rdf	37	72	57	166
onto301.rdf	15	40	0	55
onto302.rdf	16	31	0	47
onto303.rdf	56	72	0	128
onto304.rdf	41	51	1	93

Table 6.3: Number of entities in the Benchmark ontologies

6.2.2.1 Evaluation Results

In this track, *Mix'n'Match* was able to outperform all used matchers although it needed significantly more time than any other matcher to complete its mapping task (cf. Table 6.4 for overall results and Table 6.5 for detailed results). This enormous runtime is explained due to the fact that it iteratively executes its included off-the-shelf matchers

[1]http://oaei.ontologymapping.org/2012/benchmarks/index.html

and therefore accumulates their runtimes. A solution for this issue may be to run those matchers in a distributed manner, as sketched in Section 5.1.3.

Benchmark				
Matcher	Precision	Recall	F-Measure	Time
Mix'n'Match	**90,90%**	**78,81%**	**84,42%**	**196486 ms**
Anchor-Flood	89,24%	78,51%	83,53%	1360 ms
Eff2Match	87,65%	75,47%	81,11%	34657 ms
Hertuda	75,13%	55,66%	63,94%	1077 ms
HotMatch	82,46%	45,54%	58,67%	4921 ms
Aroma†	59,06%	50,15%	54,24%	1723 ms
LogMap2	84,26%	31,44%	45,79%	5141 ms

Table 6.4: Aggregated results of the Benchmark Track ranked by F-Measure

Ontologies		Precision	Recall	F-Measure
onto.rdf	onto301.rdf	91,84%	77,59%	84,11%
	onto302.rdf	87,50%	59,57%	70,89%
	onto303.rdf	86,96%	83,33%	85,11%
	onto304.rdf	97,30%	94,74%	96,00%

Table 6.5: Detailed mapping results of *Mix'n'Match* for the Benchmark track

In the Figures 6.2 to 6.5, we illustrate the progress of precision, recall and F-measure over the different mapping rounds. Based on the small size of the ontologies within this track, only a small amount of mapping iterations are necessary to complete the mapping tasks

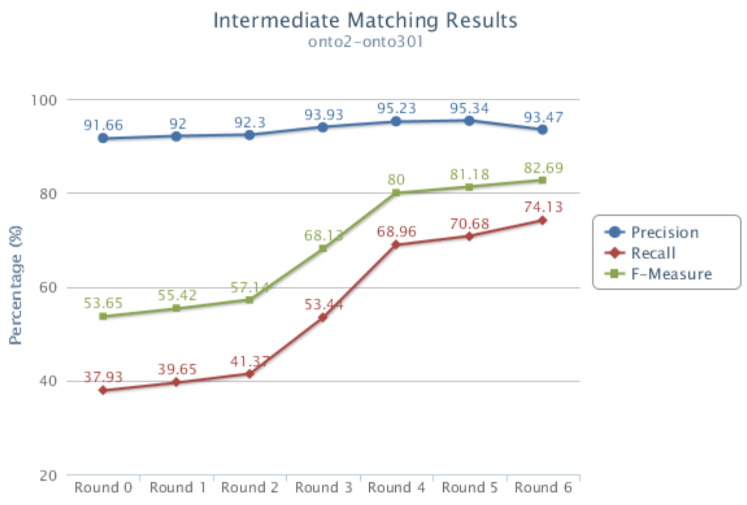


Figure 6.2: Mapping the base ontology with onto301

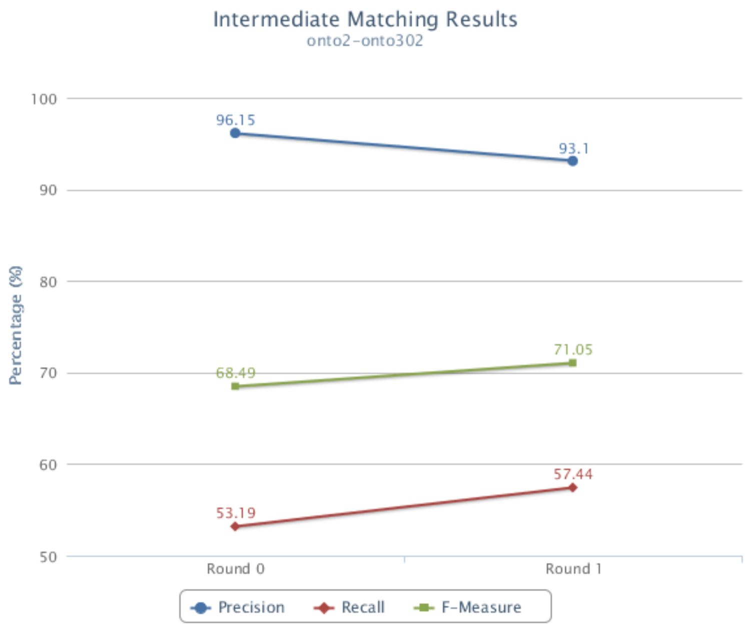


Figure 6.3: Mapping the base ontology with onto302

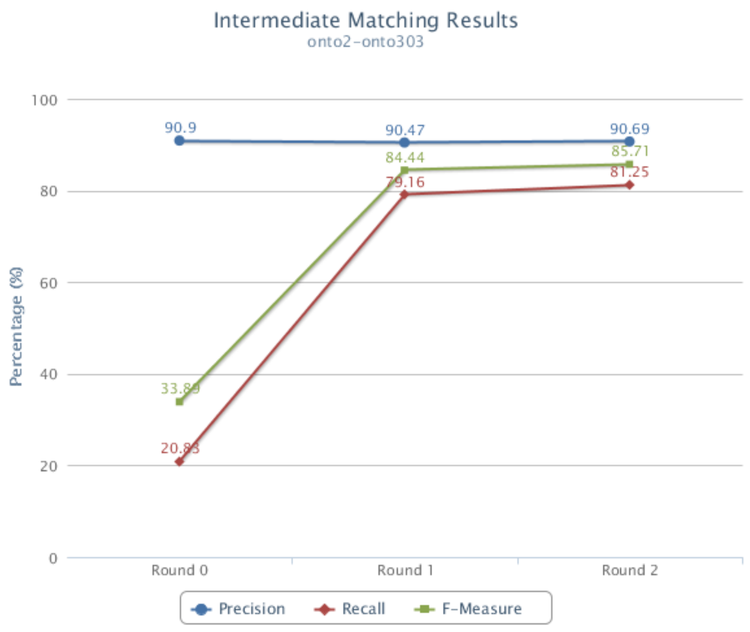


Figure 6.4: Mapping the base ontology with onto303

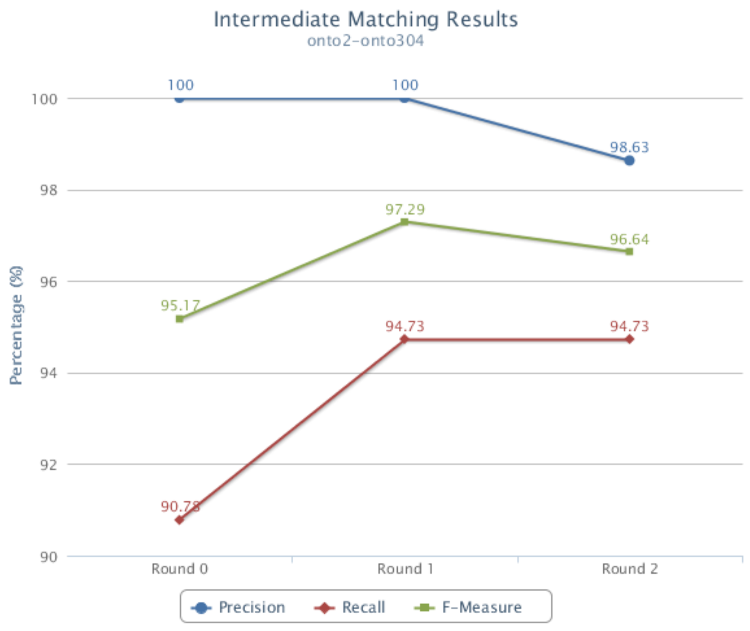


Figure 6.5: Mapping the base ontology with onto304

6.2.2.2 Comparison of Different Alignment Acceptance Thresholds

We also have investigated different alignment acceptance thresholds (cf. Section5.2.1) for the alignment selection. As we have expected, the recall value increased while the precision value decreased. But since there was a stronger decrease of precision than increase of recall, the overall F-measure value decreased too as depicted in Figure 6.6 and Table 6.6.

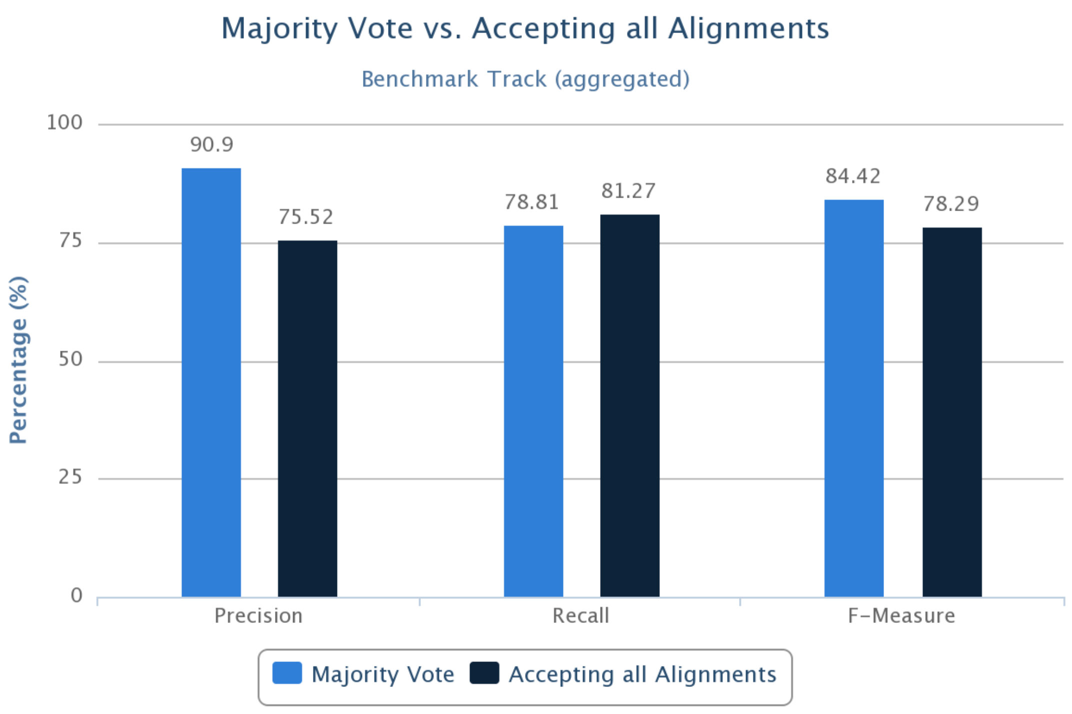


Figure 6.6: Comparing evaluation results of different alignment acceptance thresholds (overall)

Matched Ontologies		**Precision** Majority	All	**Recall** Majority	All	**F-Measure** Majority	All
onto.rdf	onto301.rdf	91,84%	78,33%	77,59%	81,03%	84,11%	79,66%
	onto302.rdf	87,50%	65,95%	59,57%	65,95%	70,89%	65,95%
	onto303.rdf	86,96%	67,80%	83,33%	83,33%	85,11%	74,77%
	onto304.rdf	97,30%	90,00%	94,74%	94,74%	96,00%	92,31%

Table 6.6: Comparing evaluation results of different alignment acceptance thresholds (detailed)

6.2.3 Conference Dataset

The Conference track[2] initially contains 16 ontologies from the same domain (conference organization), whereas only seven of them where used for our evaluations. This was mainly due to the fact, that publicly available reference alignments were only provided for those seven ontologies. Like the Benchmark track, the ontologies in this track only contain a small number of entities 6.7 and are used in real-world scenarios, but instead of mapping all ontologies against one base ontology, in this mapping task all ontologies were matched against each other as shown in Figure 6.9. Unlike the Benchmark and Anatomy tracks, this track also contains one-to-many mappings (cf. Section 3.2).

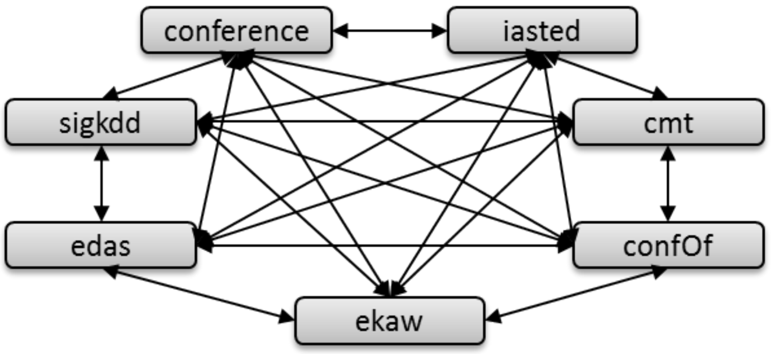


Figure 6.7: Mapping relations between the ontologies within the Conference track

Size of Conference Ontologies

Ontology Name	Number of Classes	Number of Properties	Number of Individuals	Number of Entities
cmt.owl	36	59	0	95
conference.owl	60	64	0	124
edas.owl	104	50	114	268
ekaw.owl	74	33	0	107
confOf.owl	38	36	0	74
iasted.owl	140	41	4	185
sigkdd.owl	51	28	0	79

Table 6.7: Number of entities in the Conference ontologies

[2]http://oaei.ontologymapping.org/2012/conference/index.html

6.2.3.1 Evaluation Results

As stated in Table 6.8 the overall F-measure values of the alignment results, produced by mapping the conference track, range between 35% to 67% and are worse than those retrieved by mapping the Benchmark or the Anatomy track (cf. Table 6.9 for detailed results). This is probably caused by the large majority of <1:n> mappings, which are defined in the reference alignments and the small amount of ontology matchers supporting the detection of such <1:n> mappings. Due to the current implementation of *Mix'n'Matchs* URI replacement approach, it likewise only supports the detection of <1:1> mappings, but since it was nearly able to outperform a matcher which supports the detection <1:n> mappings, we are very confident to produce even better results, after we have improved our URI replacement strategy.

Conference				
Matcher	Precision	Recall	F-Measure	Time
LogMap2	78,81%	57,91%	66,76%	22072 ms
Mix'n'Match	**71,40%**	**58,44%**	**64,27%**	**648403 ms**
Hertuda	72,62%	51,01%	59,92%	1953 ms
HotMatch	69,66%	51,60%	59,29%	8994 ms
Anchor-Flood	45,16%	57,73%	50,68%	2451 ms
Eff2Match	34,43%	64,59%	44,91%	90649 ms
Aroma	30,85%	45,97%	36,92%	5667 ms

Table 6.8: Aggregated results of the conference track ranked by F-Measure

As mentioned above, *Mix'n'Match* was able to retrieve the second highest F-Measure value of all used matchers. Nevertheless LogMap2 retrieved better results, but since our approach is primarily based on using majority vote for choosing alignments, single outstanding[3] ontology matchers don't significantly affect the results if we consider their alignments for Mix'n'Match.

We have illustrated the progress of precision, recall and F-measure over the different mapping rounds for this track too, which is depicted in Figures 6.8 to 6.14. Since the ontologies within this track contain roughly as many entities as the ones within the Benchmark track, only a small amount of mapping iterations are necessary to complete the mapping tasks.

[3]outstanding in terms of very good or very bad results

Ontologies		Precision	Recall	F-Measure
cmt	conference	41,67%	33,33%	37,04%
	confOf	71,43%	31,25%	43,48%
	edas	88,89%	61,54%	72,73%
	ekaw	85,71%	54,55%	66,67%
	iasted	57,14%	100,00%	72,73%
	sigkdd	81,82%	75,00%	78,26%
conference	confOf	69,23%	60,00%	64,29%
	edas	47,37%	52,94%	50,00%
	ekaw	58,82%	40,00%	47,62%
	iasted	71,43%	35,71%	47,62%
	sigkdd	72,73%	53,33%	61,54%
confOf	edas	56,52%	68,42%	61,90%
	ekaw	72,22%	65,00%	68,42%
	iasted	66,67%	66,67%	66,67%
	sigkdd	100,00%	57,14%	72,73%
edas	ekaw	76,47%	56,52%	65,00%
	iasted	72,73%	42,11%	53,33%
	sigkdd	87,50%	46,67%	60,87%
ekaw	iasted	63,64%	70,00%	66,67%
	sigkdd	87,50%	63,64%	73,68%
iasted	sigkdd	70,00%	93,33%	80,00%

Table 6.9: Detailed mapping results for the Conference track of *Mix'n'Match*

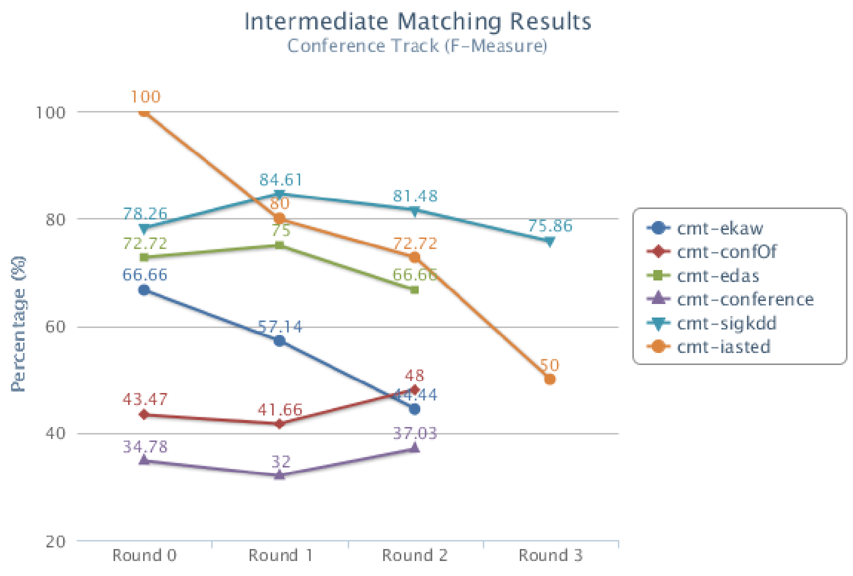


Figure 6.8: Intermediate F-measure values retrieved by mapping *cmt* against all others ontologies

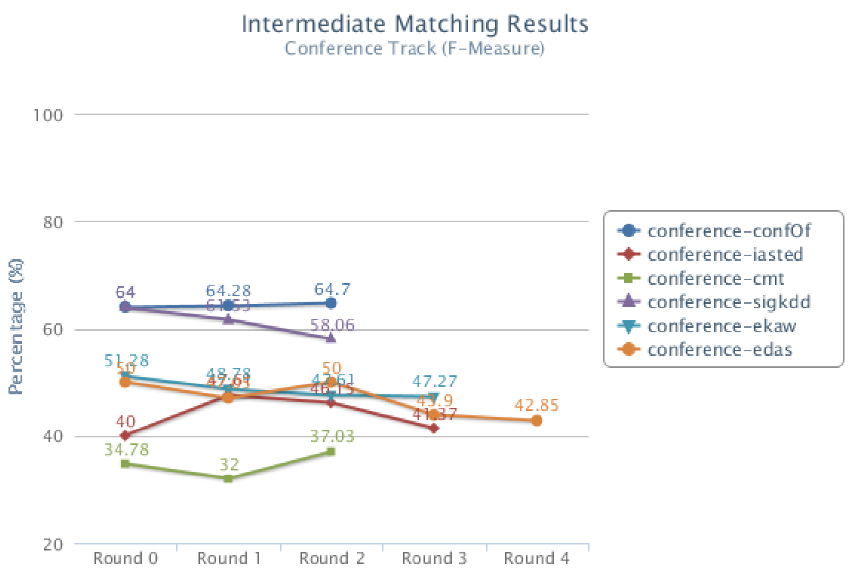


Figure 6.9: Intermediate F-measure values retrieved by mapping *conference* against all other ontologies

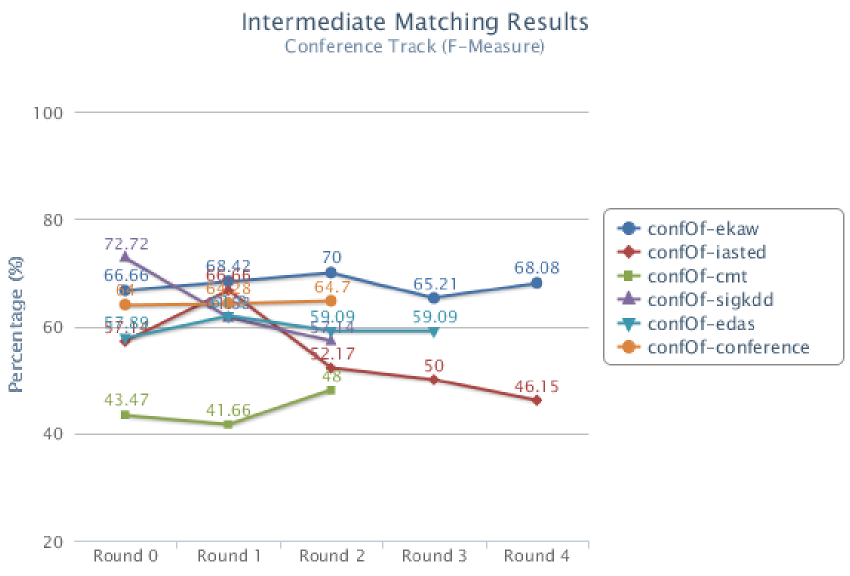


Figure 6.10: Intermediate F-measure values retrieved by mapping *confOf* against all other ontologies

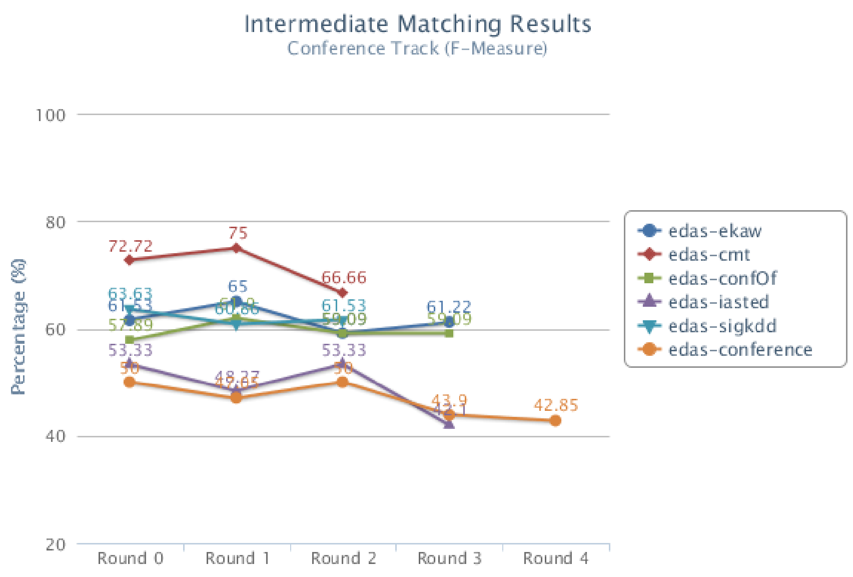


Figure 6.11: Intermediate F-measure values retrieved by mapping *edas* against all other ontologies

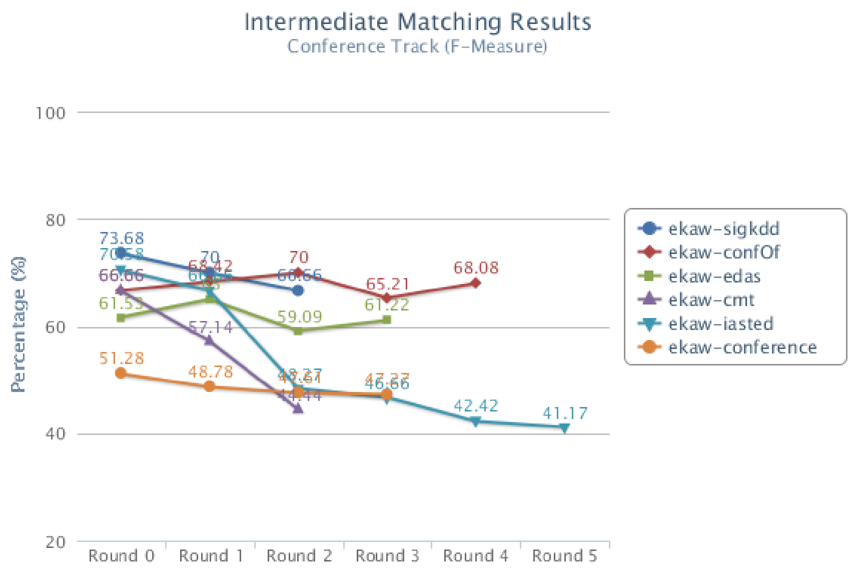


Figure 6.12: Intermediate F-measure values retrieved by mapping *ekaw* against all other ontologies

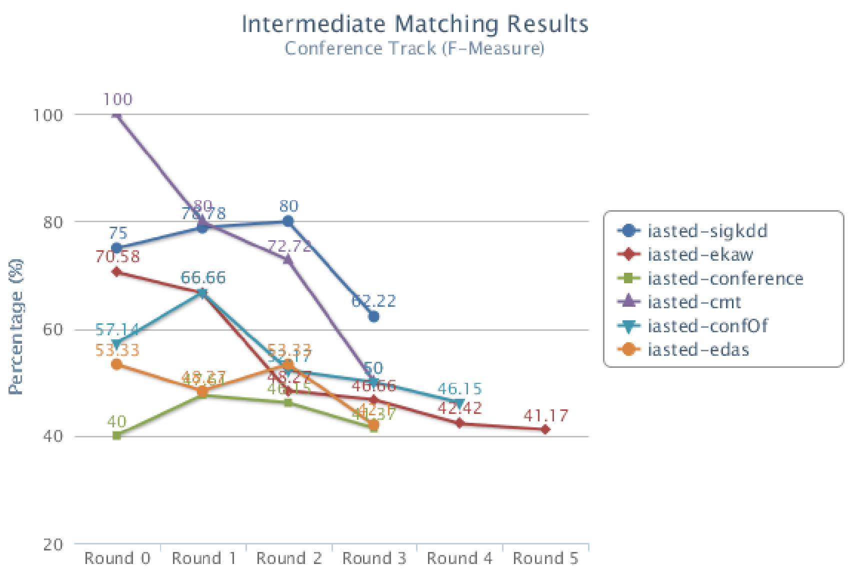


Figure 6.13: Intermediate F-measure values retrieved by mapping *iasted* against all other ontologies

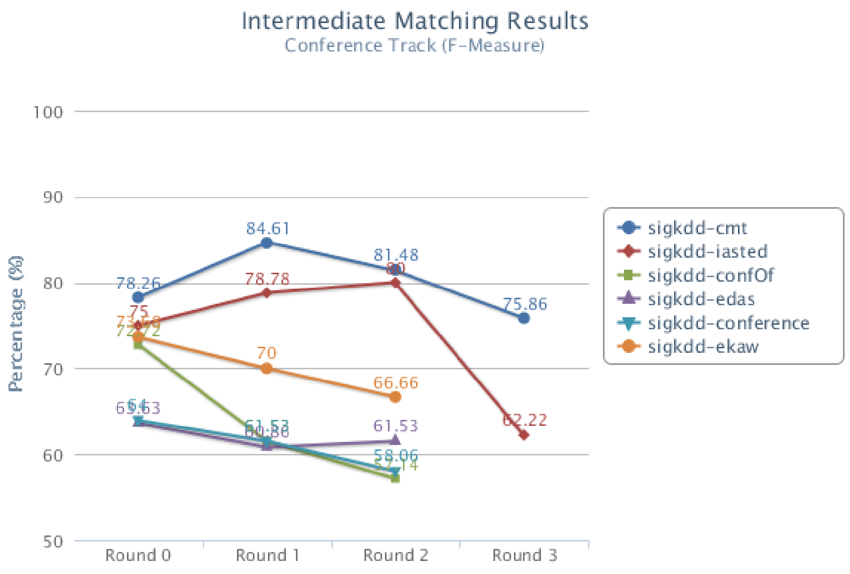


Figure 6.14: Intermediate F-measure values retrieved by mapping *sigkdd* against all other ontologies

6.2.3.2 Comparison of Different Alignment Acceptance Thresholds

By mapping the Conference track, we observed an even higher decrease of precision by accepting all alignments, in contrast to the decrease measured for the previous track. Once again, the increase of recall was not able to compensate the loss of precision, which is depicted in Figure 6.15 and Table 6.10.

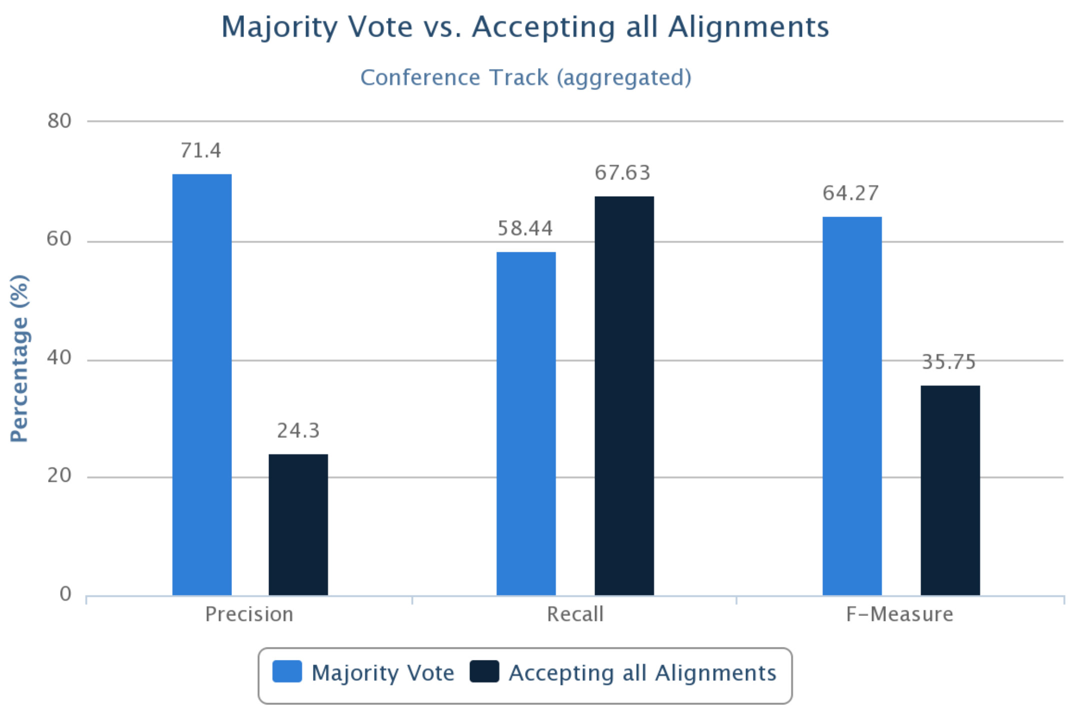


Figure 6.15: Comparing evaluation results of different alignment acceptance thresholds (overall)

Matched Ontologies		**Precision** Majority	All	**Recall** Majority	All	**F-Measure** Majority	All
cmt	conference	41,67%	17,14%	33,33%	40,00%	37,04%	24,00%
	confOf	71,43%	33,33%	31,25%	31,25%	43,48%	32,26%
	edas	88,89%	29,41%	61,54%	76,92	72,73%	42,55%
	ekaw	85,71%	16,13%	54,55%	45,45%	66,67%	23,81%
	iasted	57,14%	11,11%	100,00%	100,00%	72,73%	20,00%
	sigkdd	81,82%	37,50%	75,00%	75,00%	78,26%	50,00%
conference	confOf	69,23%	24,45%	60,00%	73,34%	64,29%	36,67%
	edas	47,37%	22,92%	52,94%	64,71%	50,00%	33,85%
	ekaw	58,82%	28,30%	40,00%	60,00%	47,62%	38,46%
	iasted	71,43%	15,38%	35,71%	57,14%	47,62%	24,24%
	sigkdd	72,73%	29,73%	53,33%	73,34%	61,54%	42,31%
confOf	edas	56,52%	28,00%	68,42%	73,68%	61,90%	40,58%
	ekaw	72,22%	44,12%	65,00%	75,00%	68,42%	55,56%
	iasted	66,67%	09,84%	66,67%	66,67%	66,67%	17,14%
	sigkdd	100,00%	13,89%	57,14%	71,43%	72,73%	23,26%
edas	ekaw	76,47%	31,38%	56,52%	69,57%	65,00%	43,24%
	iasted	72,73%	15,79%	42,11%	63,16%	53,33%	25,26%
	sigkdd	87,50%	28,57%	46,67%	66,67%	60,87%	40,40%
ekaw	iasted	63,64%	17,39%	70,00%	80,00%	66,67%	28,57%
	sigkdd	87,50%	30,43%	63,64%	63,64%	73,68%	41,18%
iasted	sigkdd	70,00%	25,45%	93,33%	93,33%	80,00%	40,00%

Table 6.10: Comparing evaluation results of different alignment acceptance thresholds (detailed)

6.2.4 Anatomy Dataset

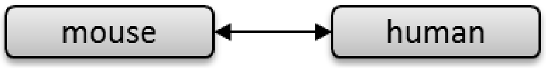


Figure 6.16: Mapping relations between the ontologies within the Anatomy track

The previous two datasets only contain quite small ontologies, therefore we extended our evaluation portfolio by the Anatomy track[4] which contains two very large ontologies 6.11 which shall be matched against each other 6.16. The aim of this track is to

[4]http://oaei.ontologymapping.org/2012/anatomy/index.html

find mappings between an adult mouse anatomy ontology and an ontology which is part of the NCI Thesaurus[5] describing the human anatomy.

Size of Anatomy Ontologies				
Ontology Name	Number of Classes	Number of Properties	Number of Individuals	Number of Entities
mouse.owl	2744	3	0	2747
human.owl	3304	2	0	3306

Table 6.11: Number of entities in the Anatomy ontologies

6.2.4.1 Evaluation Results

As Table 6.12 shows, Mix'n'Match again achieved the highest F-measure in the anatomy track. More remarkably, in Figure 6.17 we can observe the increasing F-Measure value of iterative mapping in this track. This promising results show, that (i) our approach finds additional alignments in each mapping round; (ii) was able to outperform the individual matchers used for the mapping process and (iii) since Mix'n'Match has usually a quite bad runtime performance (1400142 ms for mapping the anatomy track) but supports anytime behavior, it would have also been possible to abort the mapping process at any intermediate time receiving already found alignments. Since the increase of the F-Measure value slows down in later mapping rounds we would have been able to receive a value between 83,87%-85,95% if we would have aborted the mapping process after the half of the actually needed mapping time.

Anatomy				
Matcher	Precision	Recall	F-Measure	Time
Mix'n'Match	**94,29%**	**82,85%**	**88,2%**	1400142 ms
Logmap2	91,38%	84,63%	87,88%	16274 ms
Eff2Match	87,97%	82,98%	85,4%	85504 ms
Aroma	86,4%	68,73%	76,56%	18371 ms
AnchorFlood	87,93%	67,28%	76,23%	7073 ms

Table 6.12: Results of the anatomy track ranked by F-Measure

While we could did evaluate the effectiveness of iterative mapping described in Section 5.4 in detail, due to a lack of respective benchmarks, we have at least tried to test it within a motivating scenario. We have used two large ontologies from the anatomy track, where mapping takes considerable time each round, such that we can

[5]http://ncit.nci.nih.gov/

illustrate the effect of gathering intermediate results between mapping rounds in a reproducible fashion.

In Table 6.13 and Figure 6.17 the progression of precision, recall and F-measure value per mapping round, during the mapping of two anatomy ontologies, is shown. We were able to obtain a final F-measure value of **88,20%** after approximately 23 minutes of mapping.

Anatomy (mouse.owl - human.owl)				
mapping Round	Precision	Recall	F-Measure	Relative Increase of F-Measure
Round 0	99,42%	56,53%	72,08%	Round 0-5 +11,79%
Round 1	99,34%	59,83%	74,68%	
Round 2	99,35%	60,36%	75,09%	
Round 3	99,35%	60,49%	75,19%	
Round 4	98,75%	72,69%	83,74%	
Round 5	98,75%	72,89%	83,87%	
Round 6	97,81%	76,65%	85,95%	
Round 7	97,82%	76,85%	86,07%	Round 6-11 +2,25%
Round 8	95,46%	81,79%	88,10%	
Round 9	94,57%	82,72%	88,25%	
Round 10	94,29%	82,78%	88,16%	
Round 11	**94,29%**	**82,85%**	**88,20%**	

Table 6.13: Increasing F-Measure value after each mapping round

If we would have terminated the mapping task during the 5^{th} mapping round, we would have obtained a final F-measure value between **83,87%-85,95%**, which is quite close to the final results but only needed half of the mapping time. We have observed that the increase of F-measure value slows down in later mapping rounds, probably based on the fact that in those rounds most of the mappings have already been found and only few undetected ones were added to the final set (cf. the increase between round 0 to round 5 (**+11,79%**) and the increase between round 6 to round 11 (**+2,25%**)).

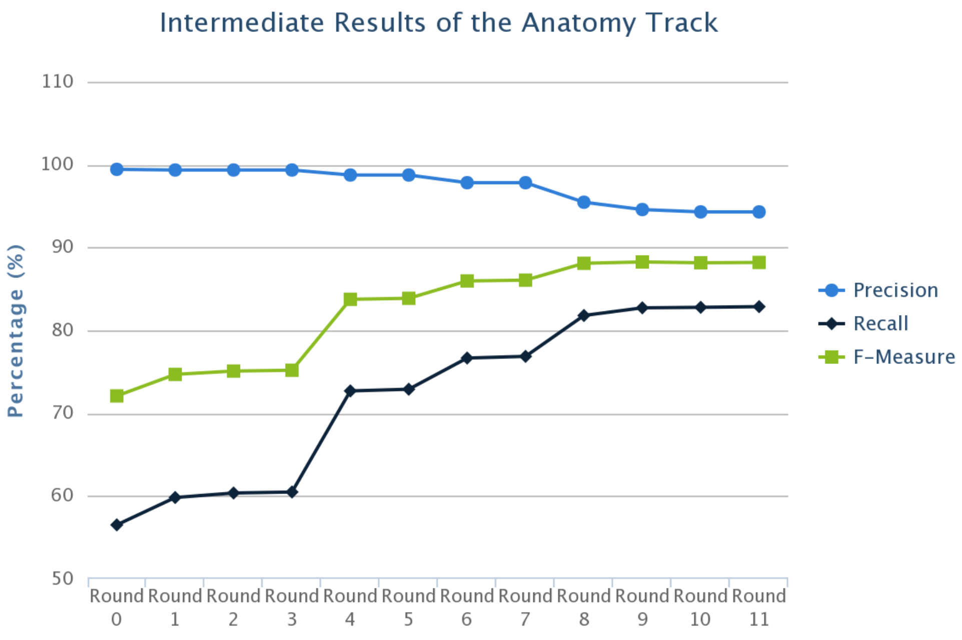


Figure 6.17: Increasing F-measure value in each mapping iteration. (anatomy track)

Although we have implemented the possibility to interrupt *Mix'n'Matchs* Mapping task at any time, we did not have defined specific constraints for the execution of such an interruption. Possible approaches to start such a termination could be for example an user interaction or a previously defined time out.

6.2.4.2 Comparison of Different Alignment Acceptance Thresholds

As we have shown for the other two tracks, Figure 6.18 indicates the differences between the two alignment acceptance thresholds. Although we would be able to increase the recall by 8%, when accepting all alignments found during mapping rounds, the precision would decrease about 24%. Nevertheless it depends on the mapping scenario and on real-world use cases, whether recall might have a higher weight than precision and therefore would make accepting all alignments more feasible or not.

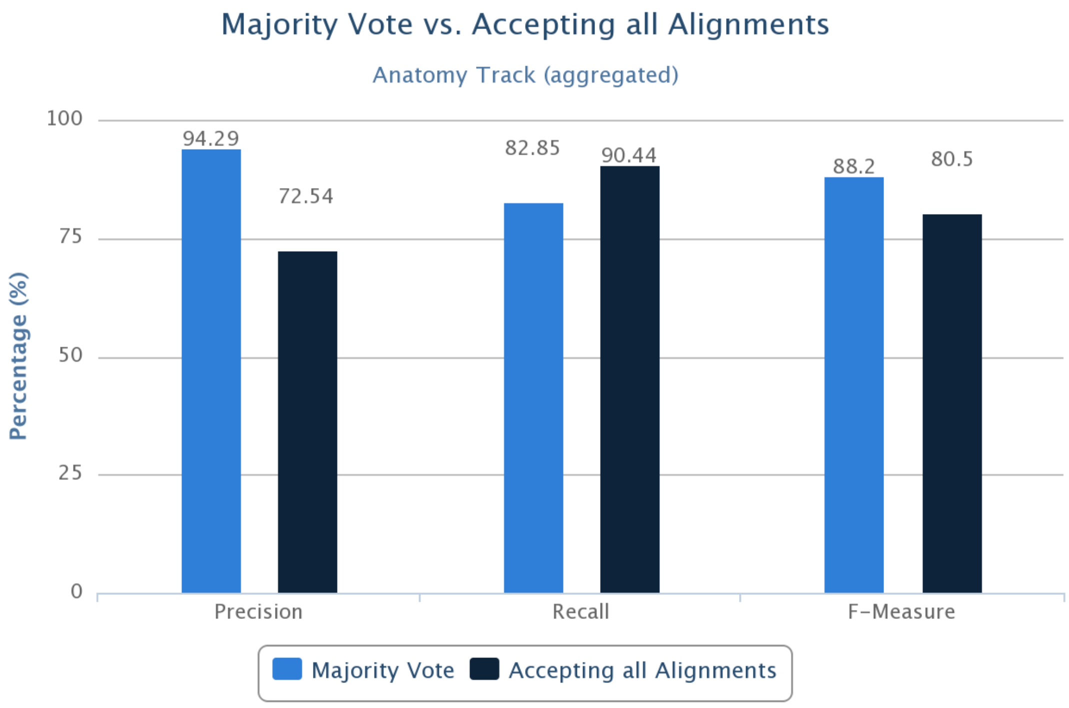


Figure 6.18: Comparing evaluation results of different alignment acceptance thresholds (overall)

6.2.5 Evaluation Conclusion

As illustrated in the present chapter, *Mix'n'Match* was able to retrieve very promising results on a selection of OAEI benchmark tracks. We were able to outperform state-of-the-art ontology matchers in two of three tested tracks and were only beaten by one matcher in the other one. Nevertheless, these evaluation results also have shown some drawbacks of our approach especially for handling <1:n> or <n:m> mapping relations (cf. results for the Conference track 6.9).

CHAPTER 7

Ontology Matching in an Industrial Environment

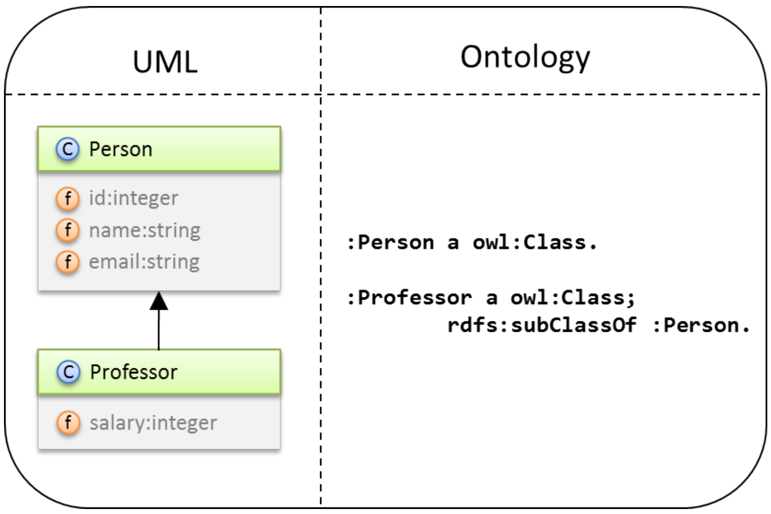
As a practical use case where we deployed our approach, in this chapter we will describe the transformation of object-oriented models, available as UML class diagrams, into ontologies by providing transformation strategies from each relevant component of UML class diagrams into its counterpart in terms of ontology elements. Afterwards we will elaborate the mapping results of *Mix'n'Match* on aligning such ontologies.

7.1 Transforming Object-Oriented Models into Ontologies

We are creating a new ontology as representation of a given object-oriented model from scratch, therefore we can define simple mapping relations between the model and the ontology. Each class in the object-oriented model corresponds to an OWL class, an attribute of a class corresponds to an OWL datatype property and an OWL object property defines a relation between classes in the given model.

In the following we will illustrate this transformations on short examples based on a sample transformation from the UML diagram given in Figure 2.8 into an ontology.

7.1.0.1 Classes and Subclasses



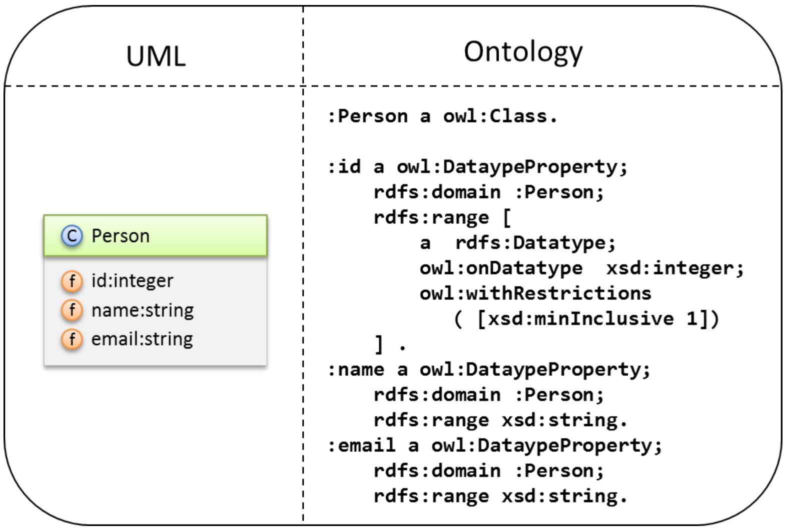
```
:Person a owl:Class.

:Professor a owl:Class;
        rdfs:subClassOf :Person.
```

Figure 7.1: Class definitions and subclass relations are transformed into OWL classes and rdfs:subClassOf relations

Classes and subclass relations are transformed straight forward, i.e. for every class and its respective subclass in the object-oriented model, classes and subclass relations in the ontology are created.

7.1.0.2 Attributes and Value Ranges

When translating datatype attributes of classes into their ontology element counterpart, it is important to maintain their value ranges and datatypes. Each datatype attribute gets transformed into an *owl:DatatypeProperty* with its *rdfs:domain* set to the class owning that attribute and a *rdfs:range* set to the respective datatype of this attribute. For any constraints which might hold for a particular attribute (uniqueness, minimum/maximum value, ...) the *rdfs:range* is set to a *blank node* which encodes the necessary restrictions. Such a sample transformation is illustrated in Figure 7.2.



```
:Person a owl:Class.

:id a owl:DataypeProperty;
    rdfs:domain :Person;
    rdfs:range [
        a  rdfs:Datatype;
        owl:onDatatype  xsd:integer;
        owl:withRestrictions
           ( [xsd:minInclusive 1])
    ] .
:name a owl:DataypeProperty;
    rdfs:domain :Person;
    rdfs:range xsd:string.
:email a owl:DataypeProperty;
    rdfs:domain :Person;
    rdfs:range xsd:string.
```

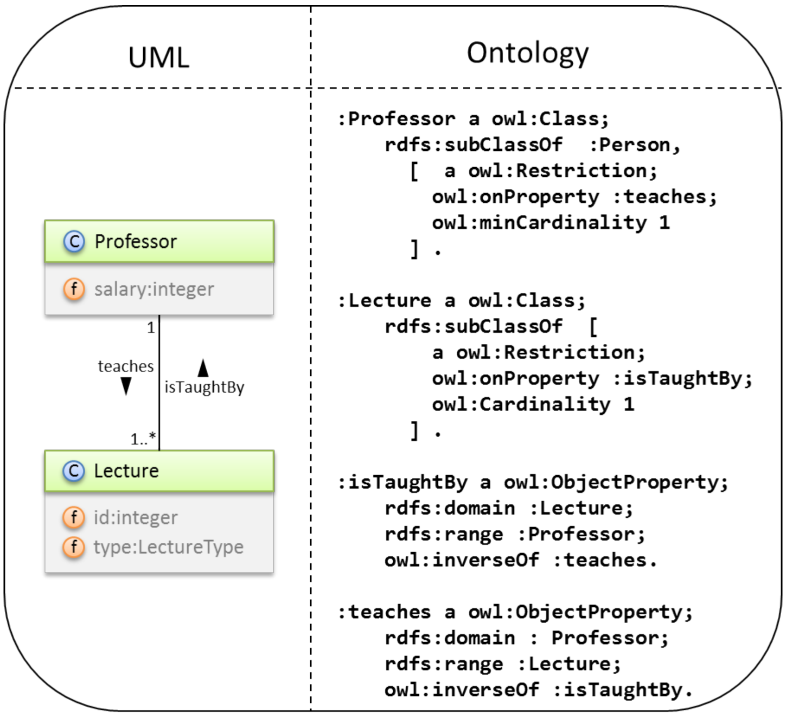
Figure 7.2: Class attributes are transformed into datatype properties with respective value ranges

7.1.0.3 Associations and Cardinalities

Another very important step of transforming UML class diagrams into ontologies, is the translation of associations between classes into *owl:ObjectProperties*. Every bi-directional association between two classes is split into two uni-directional ones and an *owl:ObjectProperty* is created for each of this associations, having its *rdfs:domain* set to the source of the association and its *rdfs:range* set to the target class. In case any cardinality constraints were defined in the UML class diagram for a specific association, a respective *owl:Restriction* on the source and target class of the association are generated as depicted in Figure 7.3.

7.1.0.4 Enumeration

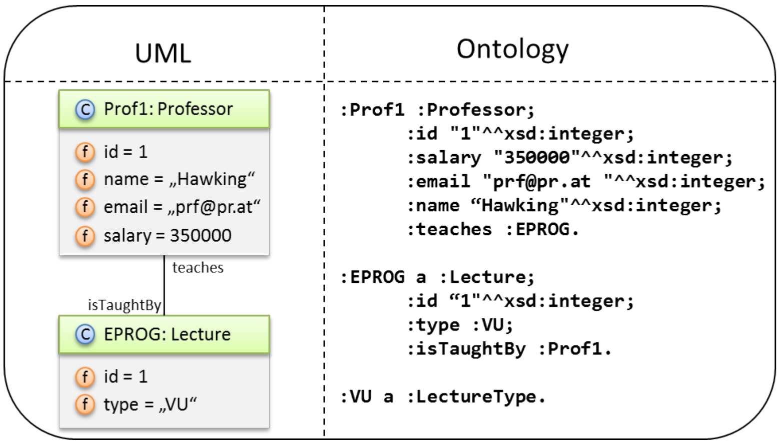
Enumerations are used to restrict the value of a specific attribute to a defined set of possible values. In Figure 7.4 we have illustrated a possible transformation strategy consisting of two major steps: (*i*) an *owl:Class* (:LectureType) is created and is defined by the *owl:oneOf* element, containing a list of objects that are its instances. A class defined by the *owl:oneOf* element contains exactly the enumerated elements, no more, no less and (*ii*) the introduction of an *owl:ObjectProperty*, which represents the fact, that a :Lecture has a specific :LectureType.



```
:Professor a owl:Class;
    rdfs:subClassOf  :Person,
      [  a owl:Restriction;
        owl:onProperty :teaches;
        owl:minCardinality 1
      ] .

:Lecture a owl:Class;
    rdfs:subClassOf  [
        a owl:Restriction;
        owl:onProperty :isTaughtBy;
        owl:Cardinality 1
      ] .

:isTaughtBy a owl:ObjectProperty;
    rdfs:domain :Lecture;
    rdfs:range :Professor;
    owl:inverseOf :teaches.

:teaches a owl:ObjectProperty;
    rdfs:domain : Professor;
    rdfs:range :Lecture;
    owl:inverseOf :isTaughtBy.
```

Figure 7.3: Relations between classes are transformed into object properties

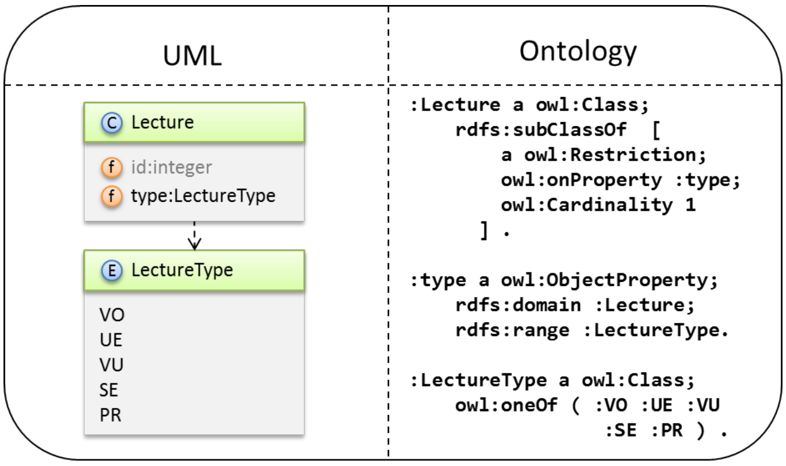
7.1.0.5 Instances



```
:Prof1 :Professor;
      :id "1"^^xsd:integer;
      :salary "350000"^^xsd:integer;
      :email "prf@pr.at "^^xsd:integer;
      :name “Hawking"^^xsd:integer;
      :teaches :EPROG.

:EPROG a :Lecture;
      :id “1"^^xsd:integer;
      :type :VU;
      :isTaughtBy :Prof1.

:VU a :LectureType.
```

Figure 7.5: Instances of objects are represented as triples using the previous defined entities

The creation of instances is straight forward. For every object which is an instantiation of a class in the UML class diagram, an individual of the respective *owl:Class* is created,



```
:Lecture a owl:Class;
    rdfs:subClassOf  [
        a owl:Restriction;
        owl:onProperty :type;
        owl:Cardinality 1
      ] .

:type a owl:ObjectProperty;
    rdfs:domain :Lecture;
    rdfs:range :LectureType.

:LectureType a owl:Class;
    owl:oneOf ( :VO :UE :VU
                :SE :PR ) .
```

Figure 7.4: Enumerations are split into an object property, an OWL class and individuals

having the same values for its attributes as the corresponding UML object.

7.2 Aligning Transformed Object-Oriented Models with Mix'n'Match

In this section we will evaluate *Mix'n'Match* on a highly industry related dataset, consisting of three object-oriented data models which were translated into ontologies.

We have practically deployed the mapping from UML models to OWL to different software systems used for configuration within Siemens in the Railway domain. While we cannot share the underlying data models and extracted ontologies, the main characteristics of these ontologies, extracted from two different systems, are shown in Table 7.1. These datasets consist themselves of several ontologies which have to be matched against each other. The characteristics of the ontologies within each dataset are described in the following.

Dataset Characteristics				
Dataset	Formalism	Relations	Language	Number of Ontologies
Railway	OWL	=, <=, !=	EN, DE	2

Table 7.1: Characteristics of the real-world dataset

Size of Railway Ontologies				
Ontology Name	Number of Classes	Number of Properties	Number of Individuals	Number of Entities
circeSW.owl	144	872	1886	166
circeZI.owl	97	567	399	55

Table 7.2: Number of entities in the railway ontologies

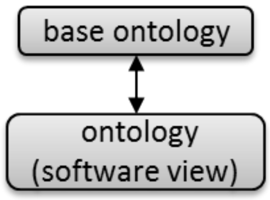


Figure 7.6: Mapping relations between the configurator ontologies

7.2.1 Evaluation Results

Due to the absence of reference alignments, we were not able to compute evaluation results like we have done for the OAEI datasets. Instead we have measured the amount of found class, property and instance alignments produced by *Mix'n'Match* and how these numbers differ from those obtained by other ontology matchers. As indicated in Figure 7.7, *Mix'n'Match* was able to detect 62 class and 171 property alignments but unfortunately no instance mappings. This is most likely caused by the majority vote, which was responsible for selecting alignments, since only one matcher (LogMap2) was able to find alignments between instances.

Although our approach was not able to find as many class and property mappings as e.g. AnchorFlood or Aroma, a manually investigation of *Mix'n'Matchs* produced alignments showed, that its set of property alignments does not contain any (or at least only very few) wrong mappings. Regarding class alignments, *Mix'n'Match* is not able to detect complex mapping relations like subsumption, yet. This issue will be targeted in future versions of *Mix'n'Match*.

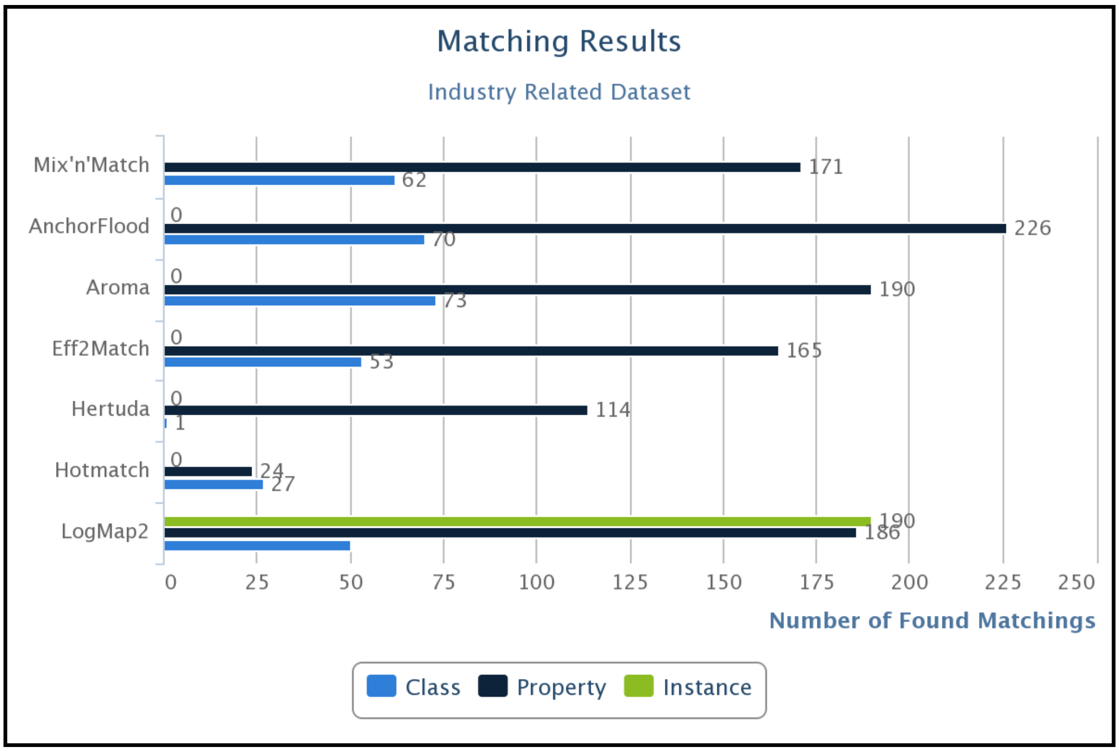


Figure 7.7: Mapping results achieved by aligning the railway ontologies

CHAPTER 8

Summary & Conclusions

In this master thesis we have presented an approach for combining off-the-shelf ontology matchers, using their combined alignment results for an iterative mapping process.

Such an approach of reusing already approved alignments as input for a new mapping task needs the support of using input alignments as additional resource for the mapping process, which is only sparsely supported by current matchers, we have shown that it is possible - to some extent - to simulate such input alignments by simply replacing the URIs of matched entities in the to be matched ontologies and then rerun the whole mapping process again. In each round we chose the alignments found by a majority of matchers as input alignments for the next rounds. Although we discovered that not all ontology matchers consider entities with the exact same URI as 100% identical, probably since string similarity measures seem to be part of nearly every investigated ontology matcher, URI replacement helped to exceed some internal confidence plateaus of found alignments and therefore accept them as correct alignments in our framework.

Furthermore we have discussed the advantages and disadvantages of several *matcher execution*, *alignment selection* and *ontology enrichment* approaches and explained why we have chosen specific strategies over others.

We have developed an architecture implementing the approach that parallelizes the iterative runs of single matchers by using multi-threading and stores found alignments in a manner that allows to stop the iterative mapping process at any time with the best alignments approved by a majority of matchers from a configurable portfolio of off-the-shelf matchers.

Overall, the results we could obtain with the *Mix'n'Match* framework, which we have implemented upon these insights and verified experimentally, are encouraging and leave various promising roads for investigations in future work.

8.1 Related Work

The approach of using an iterative process which reruns matching techniques until no further alignment pairs were found is not new per se. Tools like Anchor-Flood [44] or ASMOV [48] are based on similar frameworks but in contrast to our approach neither of them use off-the-shelf matchers for retrieving and combining alignments, but they combine several matching techniques in *one* tool. The benefit of combining whole matchers out of the box as in *Mix'n'Match*, instead of specifically implemented matching techniques is obvious: on the one hand we can highly increase the flexibility of *Mix'n'Match* since new single off-the-shelf matcher can easily be integrated into or segregated from the mapping process and on the other hand since mapping tools evolve over time and perform better and better by themselves, *Mix'n'Match* should also benefit from this evolution based on its framework.

Instead of combining ontology matchers in iterative mapping steps, some approaches only focus on the combination of alignment sets of in a one-shot process. However, as such approaches do not rerun the mapping process with the gathered additional knowledge again, we believe that they do not fully exploit the combined potential, which is somewhat confirmed by our evaluation results that needed several iterations to obtain a fixpoint. We note though that even one-shot combinations receive very good results especially when previously trained using machine learning techniques for the combination of the alignments [25, 27, 28, 56]. While we focused on unguided combination so far, a combination of machine-learning techniques for results aggregation and our iterative approach is on our agenda.

Furthermore there exist several alternatives to a simple majority vote for combining the results of different ontology matching techniques which could be fairly easily adopted to work with our approach like in [43] where the authors propose an automated approach for weighting individual ontology matchers based on their importance on different mapping tasks or in [15] were an automatic alignment evaluation by using quality measures is described.

In [16] the authors provide an approach for automatically configuring the parameters of off-the-shelf matchers and therefore optimize their performance; together with the approach proposed in [63] where unsupervised learning techniques are used to find similarity parameters, these approaches could be used to improve the performance and flexibility of *Mix'n'Match*.

8.2 Further Work

Several topics will be part of future investigations like using other combination approaches than majority vote or the distributed execution of the used ontology matchers. As alternative combination approach, an advanced machine learning technique like discussed in [27] could be used. Therefore we conjecture that we could increase the *recall* of the reference alignments generated in each round and also remove the common drawbacks of majority votes like ignoring single good results. Furthermore a self-learning classifier could be developed to ensure the full automation of such an approach.

8.2.1 Beyond Direct Mappings

Direct <1:1> mappings between two entities (atomic concepts or properties) of an ontology are the simplest way to define similarities and are supported by every ontology matcher so far. But beyond direct mapping, more complex mappings, the necessity of which was stated in [75], are not supported by the majority of current ontology matchers although several patterns for detecting complex mappings and strategies for the respective matching techniques [21,70] were proposed.

The approach presented in this thesis only considers direct <1:1> mappings between entities of ontologies; we believe the approach could be extended to support more complex mappings in terms of <1:n> and <n:m> mappings as follows: by changing the approach of naïve URI replacement for matched concepts, instead of directly replacing the URIs of these concepts, we could enrich the matched ontologies by a new introduced entities which will be equivalent to the *union* of the aligned concepts.

This workaround has to be done since combining mappings directly in a transitive fashion would result in non-conservative extensions [36] of the base ontologies, i.e. unexpected consequences could arise when using the modified ontology in place of the original one. We will illustrate this with a short example.

Listing 8.1: Ontology *O*1 before mapping

```
...
@prefix : <http://ontology.org/onto1/> .
...
:Teacher a owl:Class.
...
```

Listing 8.2: Ontology *O*2 before mapping

```
...
@prefix : <http://ontology.org/onto2/> .
...
:AssociateProfessor a owl:Class.
:FullProfessor a owl:Class.
...
```

Listing 8.3: Alignments after first round

```
...
<map>
 <Cell>
  <entity1 rdf:resource="http://ontology.org/onto1/Teacher"/>
  <entity2 rdf:resource="http://ontology.org/onto2/
     AssociateProfessor"/>
  <measure rdf:datatype="xsd:float">0.6</measure>
  <relation>=</relation>
 </Cell>
</map>

<map>
 <Cell>
  <entity1 rdf:resource="http://ontology.org/onto1/Teacher"/>
  <entity2 rdf:resource="http://ontology.org/onto2/
     FullProfessor"/>
  <measure rdf:datatype="xsd:float">0.6</measure>
  <relation>=</relation>
 </Cell>
</map>
...
```

Example 4. A possible example is shown in Listings 8.1 to 8.5; We assume here that ontology *O2*(cf. Listing 8.2) consists of two concepts, namely *FullProfessor* and *AssociateProfessor* and *O1*(cf. Listing 8.1) of one *Teacher* concept. Now, matchers may find similarities (i.e., alignments) between both, *Teacher* <-> *AssociateProfessor* and *Teacher* <-> *FullProfessor* as indicated in Listing 8.3. Unfortunately our initial approach would not be able to process these mappings, since it would only normalize the URIs from concepts in one alignment per entity found in each iteration step, say – as in our concrete current implementation – the last one, i.e. *Teacher* <-> *FullProfessor*: that behavior was implemented to ensure that the modified ontology is still a conservative extension from the unmodified one if competing alignments exists. We mean here competing in terms of URI replacement: normalizing URIs of entities within an ontology to the same URI might have unwanted side effects such as equating entities within a single ontology that were kept separate in the original design, which seems suspicious, thus not "conservative". A possible solution for keeping the integrity of the ontologies and implementing all found alignments is presented in the Listings 8.4+8.5. .

Listing 8.4: Ontology *O*1 after enrichment

```
...
@prefix : <http://ontology.org/onto1/> .
@prefix mm: <http://example.org/MixMatch/> .
...
mm:Teacher__AssociateProfessorFullProfessor rdf:type owl:Class;
owl:equivalentClass [ rdf:type owl:Class;
                      owl:unionOf ( :Teacher )
                    ] .
...
```

Listing 8.5: Ontology *O*2 after enrichment

```
...
@prefix : <http://ontology.org/onto2/> .
@prefix mm: <http://example.org/MixMatch/> .
...
mm:Teacher__AssociateProfessorFullProfessor rdf:type owl:Class;
owl:equivalentClass [ rdf:type owl:Class ;
                      owl:unionOf ( :AssociateProfessor
                                    :FullProfessor)
                    ] .
...
```

Here, we use unions of classes instead of replacing URIs directly, that is, in Listing 8.5 we in fact introduce a new local concept, which is equal to the union of the matched ones. Nevertheless the impact and applicability of this more general approach, which allows <n:m> alignments to be "emulated" by enrichment, has to be investigated in future research; similar considerations as for using `owl:equivalentClass` mentioned in Chapter 5 above may arise, in that such modeled "enriched alignments" would not necessarily considered properly by many matchers.

8.2.2 Distributed Computation

As already discussed in Chapter 5, another very important part of future investigations will be the usage of distributed computation techniques for the mapping process and combination of matchers as shown in Figure 5.4. Since *Mix'n'Match* relies on the results received by other ontology mapping tools, the respective time necessary for *Mix'n'Match* to complete its alignment process is much higher than that of its used mapping tools, even by using a local multi-threading approach for the matcher execution instead of calling them sequentially. Nevertheless the runtime of the multi-threaded approach was quite better than the one produced by the sequential one.

It seems an obvious and promising extension to embed our approach in distributed computation concepts like cloud computing, were we would be able to outsource the individual ontology matchers into separate cloud processes and therefore avoid integrating issues like concurrent libraries in the build path as well as may be able to further decrease the runtime of *Mix'n'Match*.

Bibliography

[1] Rakesh Agrawal, Alexander Borgida, and HV Jagadish. *Efficient management of transitive relationships in large data and knowledge bases*, volume 18. ACM, 1989.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

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

[4] Dave Beckett and Brian McBride. RDF/XML syntax specification (revised). *W3C recommendation*, 10, 2004.

[5] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. Turtle – Terse RDF Triple Language. W3C Candidate Recommendation, February 2013. `http://www.w3.org/TR/2013/CR-turtle-20130219/`.

[6] Tim Berners-Lee and Dan Connolly. Notation3 (N3): A readable RDF syntax. *W3C Team Submission (January 2008) http://www. w3. org/TeamSubmission*, (3), 1998.

[7] Olivier Bodenreider. The unified medical language system (UMLS): integrating biomedical terminology. *Nucleic acids research*, 32(suppl 1):D267–D270, 2004.

[8] Paolo Bouquet, Luciano Serafini, and Stefano Zanobini. Semantic coordination: a new approach and an application. In *The Semantic Web-ISWC 2003*, pages 130–145. Springer, 2003.

[9] Steve Bratt. Semantic Web, and Other Technologies to Watch. W3C, 2007. `http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#%2824%29`.

[10] Dan Brickley. RDF: Understanding the Striped RDF/XML Syntax, 2002. `http://www.w3.org/2001/10/stripes/`.

[11] Dan Brickley and Ramanathan V Guha. RDF vocabulary description language 1.0: RDF schema. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-schema/`.

[12] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, 2006.

[13] Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis. On labeling schemes for the semantic web. In *Proceedings of the 12th international conference on World Wide Web*, pages 544–555. ACM, 2003.

[14] Watson Wei Khong Chua and Jung-Jae Kim. Eff2Match results for OAEI 2010. In *OM*, volume 689 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[15] Isabel F Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. Efficient selection of mappings and automatic quality-driven combination of matching methods. In *ISWC International Workshop on Ontology Matching (OM) CEUR Workshop Proceedings*, volume 551, pages 49–60. Citeseer, 2009.

[16] Isabel F Cruz, Alessio Fabiani, Federico Caimi, Cosmin Stroe, and Matteo Palmonari. Automatic configuration selection using ontology matching task profiling. In *The Semantic Web: Research and Applications*, pages 179–194. Springer, 2012.

[17] Thanh Tung Dang et al. HotMatch results for OEAI 2012. In *Seventh International Workshop on Ontology Matching (OM 2012)*, 2012.

[18] Jérôme David, Fabrice Guillet, and Henri Briand. Matching directories and OWL ontologies with AROMA. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pages 830–831, New York, NY, USA, 2006. ACM.

[19] J Dean and S Ghemawat. MapReduce: simplified data processing on large clusters. *Commun ACM*, 51:107–113, 2008.

[20] Thomas L Dean and Mark S Boddy. An Analysis of Time-Dependent Planning. In *AAAI*, volume 88, pages 49–54, 1988.

[21] Robin Dhamankar, Yoonkyong Lee, Anhai Doan, Alon Halevy, and Pedro Domingos. iMAP: discovering complex semantic matches between database schemas. In *in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, ACM*. Press, 2004.

[22] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, pages 221–237. Springer, 2003.

[23] Hong-Hai Do and Erhard Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.

[24] Anhai Doan and Alon Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 26(1):83–94, 2005.

[25] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal—The International Journal on Very Large Data Bases*, 12(4):303–319, 2003.

[26] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In *Handbook on ontologies*, pages 385–403. Springer, 2004.

[27] Kai Eckert, Christian Meilicke, and Heiner Stuckenschmidt. Improving Ontology Matching Using Meta-level Learning. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC 2009 Heraklion, pages 158–172, Berlin, Heidelberg, 2009. Springer-Verlag.

[28] Marc Ehrig, York Sure, and Staab Steffen. Bootstrapping Ontology Alignment Methods with APFEL. In *Proceedings of the 4th International Semantic Web Conference (ISWC)*. Springer, November 6-10 2005.

[29] Jérôme Euzenat. Towards composing and benchmarking ontology alignments. In *Proc. ISWC-2003 workshop on semantic information integration, Sanibel Island (FL US)*, pages 165–166. Citeseer, 2003.

[30] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn dos Santos. Ontology Alignment Evaluation Initiative: Six Years of Experience. In *Journal of Data Semantics XV*, volume 6720 of *Lecture Notes in Computer Science*, pages 158–192. Springer, Berlin, Heidelberg, 2011.

[31] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, 2007.

[32] Jerome Euzenat and Petko Valtchev. Similarity-based ontology alignment in OWL-Lite. In *ECAI 2004: 16th European Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: Including Prestigious Applicants [sic] of Intelligent Systems (PAIS 2004): Proceedings*, volume 110, page 333. IOS Press, 2004.

[33] Jérôme Euzenat et al. State of the art on ontology alignment. *Knowledge Web Deliverable D2.2.3*, 2:2–3, 2004.

[34] Dieter Fensel. *Ontologies:: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, 2001.

[35] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Languange*. Addison-Wesley Professional, 2004.

[36] Silvio Ghilardi, Carsten Lutz, and Frank Wolter. Did I Damage my Ontology? A Case for Conservative Extensions in Description Logics. In *In Proc. of KR2006*, pages 187–197. AAAI Press, 2006.

[37] Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. 2003.

[38] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. *S-Match: an algorithm and an implementation of semantic matching*. Springer, 2004.

[39] Object Management Group. Unified Modeling Language 2.0 Superstructure Specification. `http://www.omg.org/spec/UML/2.0/Superstructure/PDF/`, 2005. [Online; accessed 19-September-2013].

[40] OWL Working Group. OWL 2 Web Ontology Language. *W3C recommendation*, (2012-12), 2012.

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

[42] SKOS Core Guide, Eds A Miles, and D Brickley W3C. SKOS Core Guide. *Work*, 501:34049.

[43] Marko Gulić, Ivan Magdalenić, and Boris Vrdoljak. Automated weighted aggregation in an ontology matching system. *International Journal of Metadata, Semantics and Ontologies*, 7(1):55–64, 2012.

[44] Md. Seddiqui Hanif and Masaki Aono. An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *J. Web Sem.*, 7(4):344–356, 2009.

[45] Patrick Hayes and Brian McBride. RDF semantics. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-mt/`.

[46] Sven Hertling. Hertuda results for OEAI 2012. In *Seventh International Workshop on Ontology Matching (OM 2012)*, 2012.

[47] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member submission*, 21:79, 2004.

[48] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology Matching with Semantic Verification. *Web Semantics*, 7(3):235–251, September 2009.

[49] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Yujiao Zhou. LogMap 2.0: towards logic-based, scalable and interactive ontology matching. In *Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences*, SWAT4LS '11, pages 45–46, New York, NY, USA, 2012. ACM.

[50] Yannis Kalfoglou and W. Marco Schorlemmer. Ontology mapping: the state of the art. *Knowledge Eng. Review*, 18(1):1–31, 2003.

[51] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. ELK reasoner: Architecture and evaluation. In *Proc. of the OWL Reasoner Evaluation Workshop (ORE'12)*, volume 858, 2012.

[52] Graham Klyne, Jeremy J Carroll, and Brian McBride. Resource description framework (RDF): Concepts and abstract syntax. *W3C recommendation*, 10, 2004.

[53] Ora Lassila and Ralph R Swick. Resource description framework (RDF) model and syntax specification. 1999.

[54] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[55] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. Mafra—a mapping framework for distributed ontologies. In *Knowledge engineering and knowledge management: ontologies and the semantic web*, pages 235–250. Springer, 2002.

[56] Paulo Maio, Nuno Bettencourt, Nuno Silva, and João Rocha. Evaluating a Confidence Value for Ontology Alignment. In *Proceedings of the 2nd International Workshop on Ontology Matching (OM-2007) Busan, Korea, November 11, 2007*, volume 304 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[57] Deborah L McGuinness, Frank Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10(2004-03):10, 2004.

[58] Alistair Miles and Dan Brickley. SKOS core vocabulary specification. *W3C working draft*, 2, 2005.

[59] George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[60] Malgorzata Mochol and Anja Jentzsch. Towards a Rule-Based Matcher Selection. In *Proceedings of the 16th international conference on Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes in Computer Science*, pages 109–119. Springer, 2008.

[61] Victoria Nebot and Rafael Berlanga. Efficient retrieval of ontology fragments using an interval labeling scheme. *Information Sciences*, 179(24):4151–4173, 2009.

[62] Duy Hoa Ngo and Zohra Bellahsene. YAM++ : (not) Yet Another Matcher for Ontology Matching Task. In *Bases de Données Avancées*, page 5, France, 2012.

[63] Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta. Unsupervised learning of link discovery configuration. In *The Semantic Web: Research and Applications*, pages 119–133. Springer, 2012.

[64] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9. ACM, 2001.

[65] Natalya F Noy and Mark A Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

[66] Bijan Parsia and Evren Sirin. Pellet: An owl dl reasoner. In *Third International Semantic Web Conference-Poster*, page 18, 2004.

[67] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. WordNet:: Similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004*, pages 38–41. Association for Computational Linguistics, 2004.

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

[69] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[70] Dominique Ritze, Christian Meilicke, Ondrej Svab-Zamazal, and Heiner Stuckenschmidt. A Pattern-based Ontology Matching Approach for Detecting Complex Correspondences. In *Fourth International Workshop on Ontology Matching (OM-2009)*, volume 551 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[71] Cornelius Rosse and José LV Mejino Jr. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *Journal of biomedical informatics*, 36(6):478–500, 2003.

[72] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[73] Balthasar Schopman, Shenghui Wang, Antoine Isaac, and Stefan Schlobach. Instance-based ontology matching by instance enrichment. *Journal on Data Semantics*, 1(4):219–236, 2012.

[74] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *OWLED*, volume 432, 2008.

[75] P. Shvaiko and J. Euzenat. Ontology Matching: State of the Art and Future Challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):158–176, 2013.

[76] Nuno Silva and Joao Rocha. Multidimensional service-oriented ontology mapping. *International journal of Web engineering and technology*, 2(1):50–80, 2005.

[77] Simon Steyskal and Axel Polleres. Mix'n'Match: An Alternative Approach for Combining Ontology Matchers. In *Proceedings of the 12th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2013)*, Graz, Austria, September 2013.

[78] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. A string metric for ontology alignment. In *The Semantic Web–ISWC 2005*, pages 624–637. Springer, 2005.

[79] Edward Thomas, Jeff Z Pan, and Yuan Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *The Semantic Web: Research and Applications*, pages 431–435. Springer, 2010.

[80] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Automated reasoning*, pages 292–297. Springer, 2006.

[81] Mike Uschold. Achieving semantic interoperability using RDF and OWL - v4, 2005. `http://lists.w3.org/Archives/Public/public-swbp-wg/2005Sep/att-0027/SemanticII-v4.htm` Last accessed on 13.09.2013.

[82] William E Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.

[83] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996.

[84] Antoine Zimmermann, Ratnesh Sahay, Ronan Fox, and Axel Polleres. Heterogeneity and context in semantic-web-enabled HCLS systems. In *On the Move to Meaningful Internet Systems: OTM 2009*, pages 1165–1182. Springer, 2009.