Generating Complex Ontology Alignments

Bachelor Thesis

presented by
Dominique Ritze
Matriculation Number 1067187

submitted to the
Lehrstuhl für Künstliche Intelligenz
Prof. Dr. Heiner Stuckenschmidt
University Mannheim

May 2009

Contents

1 Introduction **1**
1.1 Ontology Matching 2
1.2 Problem Statement 6
1.3 Related Work 9
1.4 Outline and Contribution 11

2 Exploratory Study **12**
2.1 Approach 12
2.2 Types of Complex Correspondences 13

3 Algorithms **23**
3.1 Unqualified Restriction 27
3.2 Qualified Restriction 30
3.3 Value Restriction 30
3.4 Property chain 33

4 Experiments **39**
4.1 Implementation 39
4.2 Settings 44
4.3 Results 46

5 Summary **50**
5.1 Conclusion 50
5.2 Future Work 51

A Program Code / Resources **56**

List of Algorithms

1 Unqualified Restriction 28
2 Qualified Restriction 31
3 Value Restriction 34
4 Property Chain 37

List of Figures

1.1 Matching Approaches 4
1.2 Example complex correspondence 8

2.1 Types of complex correspondences 14

3.1 Example subclass hierarchy in two ontologies 25
3.2 Unqualified Restriction example 29
3.3 Qualified Restriction example 32
3.4 Value Restriction example 35
3.5 Property Chain example 38

4.1 Implementation overview 43

List of Tables

2.1 Number of types 20

4.1 Considered conference ontologies 44
4.2 Results low thresholds 47
4.3 Results mid thresholds 47
4.4 Results high thresholds 48

Chapter 1

Introduction

Nowadays a huge amount of information is available via the World Wide Web. Everyone can distribute data and a lot of companies sell their goods online. This leads to several problems especially to find information a user is looking for. The reason behind is that computers, unlike humans, cannot really gather the facts and particularly not understand them. Also the decentralized organization of the Internet does not help to solve this problem, in contrast, it supports the heterogeneity of information. Due to the facts mentioned above it is not possible to get all information and to represent them in a consistent way. The Semantic Web, introduced by Tim Berners-Lee 2001 [2], tries to find a solution by constituting all information in a machine-readable way. Discovering and integrating information and additionally infer new data out of given data are the main Semantic Web challenges. To achieve the ideas some standards are necessary to have a common foundation. Therefore XML (Extensible Markup Language) [3], as a standard of data exchange, the languages RDF [12] (Resource Description Framework) and OWL [16] (Web Ontology Language) are implemented and recommended by the W3C (World Wide Web Consortium) [1]. RDF, especially RDF-Schema, a layer on top of RDF, and OWL are so-called ontology languages to qualify ontologies. An ontology contains information about a particular domain of interest in a machine-readable way. There are no predefined rules how to create an ontology or which vocabulary to use. Hence anyone can create an ontology in their own way and the ontologies can differ a lot although they describe the same issue or domain. That is why integrating or comparing ontologies is not as easy as it might seem. Ontology matching tries to find solutions for integrating heterogeneous ontologies and is a fundamental requirement for achieving the vision of the Semantic Web.

[1] http://www.w3c.com

1.1 Ontology Matching

In this section ontology matching and its components are formalized. After giving a first overview, the existing ontology matching techniques and strategies are explained. Ontology matching is a large area of research and there are conferences which only deal with this topic such as the OAEI (Ontology Alignment Evaluation Initiative) [4]. Additionally on some other conferences papers about ontology matching are presented, like on the ISWC (International Semantic Web Conference) [2]. Moreover many journals pick up the topic, such as the Journal on Data Semantics. About thirty matching systems show the relevance of ontology matching, Falcon [10], SAMBO [14], CIDER [8], ASMOV [11] to mention just a few. For further information about conferences, journals or any other kind, see [3]. A whole book with about 300 pages called "Ontology Matching" written by Euzenat and Shvaiko [6] only concentrates on ontology matching. Some definitions, as suggested by [6], are needed for further explanations:

Definition 1 *An ontology is a tuple* $o =< C, I, R, T, V, \leq, \perp, \in, => $ *such that:*

- C *is a set of classes*
- I *is a set of individuals*
- R *is a set of relations*
- T *is a set of data*
- V *is a set of values*
- $\leq$ *is a relation on* $(C\times C) \cup (R\times R) \cup (T\times T)$ *called specialization*
- $\perp$ *is a relation on* $(C\times C) \cup (R\times R) \cup (T\times T)$ *called exclusion*
- $\in$ *is a relation over* $(I\times C) \cup (V\times T)$ *called instantiation*
- $=$ *is a relation over* $I \times R \times (I\cup V)$ *called assignment*

Classes are the main entities and represent a set of individuals in the domain. Every individual is a particular component and belongs to a class. All relations are functions and relate a class to another class or a datatype. Especially in OWL the relations are divided in two groups: object properties and datatype properties. Object properties have the range classes, datatype properties map classes to

[2] http://iswc2008.semanticweb.org/
[3] http://ontologymatching.org/

datatypes such as string or integer. Values are just used to express individuals, for example to define the name of an individual. Specialization conforms to the concept of generalization. A class/relation/datatype can be more general/specific than another class/relation/datatype. Imagine a class called *Person* and another class *Man*. *Man* $\leq$ *Person* means that every *Man* is also a *Person* but not the other way round. OWL calls $\leq$ *subClassOf/subPropertyOf* and $\geq$ which is not necessary but often advantageous *superClassOf/superPropertyOf.* Exclusion describes disjoint classes/properties, having intersection empty, in OWL the exclusion is introduced by *disjointWith*. Allocating an individual to its corresponding class is called instantiation. The last component of an ontology is called assignment and relates an individual to another one or to a value by a certain property. Imagine an individual of class *Human* and this individual should be related to a string which express its last name. Additionally a datatype property *lastName* is available which relates a human to its last name. Referring the three parts: individual, property, here *lastName*, and a value, here an arbitrary string, is the task of the assignment relation. After explaining all necessary concepts of an ontology they are available for further definitions.

Definition 2 *Given two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, Q a function defining a set of matchable elements, a set of semantic relations R and a confidence structure D. A correspondence is a 4-tuple: $< e, e', r, n >$ such that:*

- $e \in Q(\mathcal{O}_1)$
- $e' \in Q(\mathcal{O}_2)$
- $r \in R$
- $n \in D$

Mostly e and e' are entities of the ontologies such as classes, properties or individuals. The semantic relation r between e and e' is often confined to the equivalence relation, but is not restricted to the equivalence relation in general. A confidence structure is an ordered set of degrees which contains a smallest and a highest element. Usually n is a numerical value in between 0.0 and 1.0. and can be seen as a measure of trust in a conceivable correspondence. To illustrate the definition, a simple example for a correspondence between two classes with an equivalence relation:
http://foo.com/ontology1#Person = http://foo.com/ontology2#Human.
The example declares that every *Person* is also a *Human* and vice versa. Because of having a full confidence no further declaration is given. For identifying entities their names are represented by URIs (universal resource identifier).

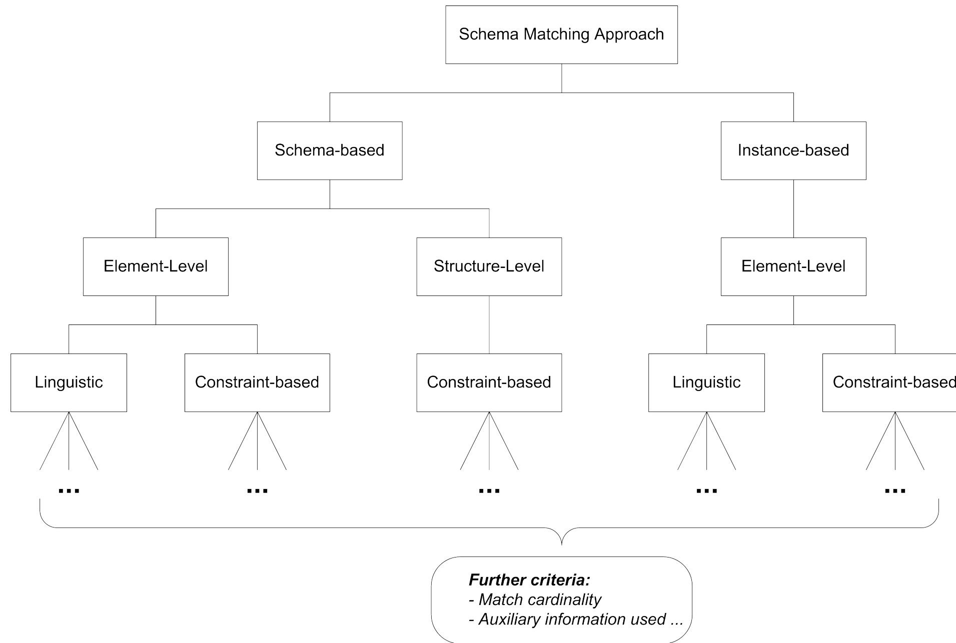

Figure 1.1: Overview of matching approaches divided into several levels

Definition 3 *Given two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ an alignment is a set of correspondences.*

Now all needed definitions are given and ontology matching can be defined. Ontology matching is the process of finding all correct correspondences between two ontologies. Naturally not every correspondence is found and not every one is necessarily correct but both are aspired.

After all important terms have been defined, existing techniques and approaches for ontology matching are explained. A short overview of several ontology matching strategies is shown in Figure 1.1 based on the figure in [19].

On the first level the strategies are separated into schema-based strategies on the left and instance-based strategies on the right. Schema-based systems only use information about the ontology schema and not, differently to instance-based systems, information about instances. Not every ontology necessarily contains instances hence the instance-based strategy cannot always be applied.
The second level shows the partition into element-level and structure-level policies. While element-level matching systems only consider entities isolated to find

a correspondence, e.g. town = city, structure-level systems analyze combinations of entities, e.g. sister = sibling with hasGender female. Instance-based matching can only be element-based because no structure for individuals exists.
Next level divides matching strategies into linguistic and constraint-based approaches. Linguistic approaches base on names and texts of the entities to find correspondences. There are several possibilities where additional information such as dictionaries are not required, for example to compare names, prefixes, suffixes, substrings or to compute special similarity metrics like the hamming distance. But it is also possible to use thesauri or dictionaries in order to find synonyms, e.g. person = human, hypernyms, e.g. animal is a hypernym of dog, generalization or names in other languages. Ontologies often include constraints such as restrictions on domain or range of a relation, e.g. not every relation can link an instance to another, or cardinalities, how many instances of a class can be part of one relation. Also the class and relation hierarchy can be used to draw conclusions. Only element-level approaches can apply linguistic conceptions because they compare isolated entities with an obvious, existent name instead of structure-level.
Further criteria can distinguish various approaches such as the matching cardinality or auxiliary information. The matching cardinality determines how many entities of an ontology can match how many entities out of another ontology. In general three options are feasible: 1:1, 1:n, n:m. 1:1 means every entity can only match at most one entity, 1:n one entity can match arbitrary entities of the other ontology and n:m claims no restriction at all.

Furthermore taking the decision which candidates are correspondences is not always easy. There might be various possibilities and especially if a 1:1 approach is chosen, a matching system has to decide which truly is a correspondence. Often a certain threshold is determined and every similarity value below this boarder is not examined any further.

Making use of auxiliary information, such as dictionaries, thesauri, user-provided data or the reuse of available alignments can help to find new correct correspondences and to avoid incorrect ones.

To get better results it is of course possible to combine several approaches. Therefore two main suggestions are submitted: hybrid and composite matcher. A hybrid matcher just integrates a few defined strategies and executes them in a fixed order. Composite matcher join the results of different approaches and allow to change them or the executing. At the end the values have to be summarized to get just one similarity value for a pair of entities.

Beside all these techniques, learning methods can also be applied. Bayes learning, neural networks and decision trees are machine learning approaches which can

be used to match ontologies. Differently to other techniques, learning methods require sample data to learn from. If no data is available it is not possible to create alignments with learning methods.

Finally a matcher can compute the correspondences either fully automatically or with user interaction to take decisions. Fully automatic matchers usually do not provide such good results as matchers with user interaction because humans can delete incorrect correspondences just by regarding suggestions. Having a large number of techniques allows the best matcher to be chosen individually for two given ontologies. If a variety may be best in one case, in another this might not be the same. Every strategy has advantages and disadvantages and one cannot select the best approach without considering the ontologies.

There are a lot of strategies but ordinarily only equivalent relations just between two entities are realized. Complex correspondences are not at present considered and established matching systems rarely support them.

1.2 Problem Statement

After summarizing ontology matching in general and the existing techniques in the previous section, disadvantages and issues of these strategies are explained in this section. The need for generating complex alignments becomes reasonable if ontologies contain complex correspondences which can be found by humans but not by matching systems.

Generating complex correspondences is useful because everyone can create an ontology in their own way. Different variations can arise for several reasons which are not mandatorily avoidable. Three fundamental reasons for heterogeneity are the target goal, usage of an ontology and also the background knowledge about the domain. These problems cannot be avoided and even if all three issues could be resolved, the ontologies are naturally not exactly the same. Additionally other causes, which are described further, raise heterogeneity. The following classification is based on the heterogeneity types in the ontology matching book [6]. Types of heterogeneity are:

Syntactic heterogeneity An essential difference between ontologies can be at the syntactical level. If they are expressed in various ontology languages they are not syntactically the same and it is often not possible to compare them without converting one ontology into another language.

Terminological heterogeneity Every object can have names in different languages or it can also be a synonym, having different names for the same meaning. In this case, a comparison cannot be executed without additional information such as a dictionary or thesaurus.

Conceptual heterogeneity The conceptual heterogeneity basically describes the problem of modeling the same issue in arbitrary ways although there are no syntactical or terminological problems. To get a more precise concept, a separation into three parts can be made.

Difference in coverage Gap between describing different issues, e.g. describing car and animal.

Difference in granularity The same parts are explained but the level of details does not agree. For example one can describe the world on a very low level and define every atom or one can characterize humans, animals and landscape on a very high level. Both characterize the same world but the descriptions are not comparable.

Difference in perspective Although coverage and granularity are similar, a problem can occur because of looking at the domain from different perspectives. Everyone has its own perspective of a given circumstance or object, e.g. some specialist on computer science has a technical view of a computer different from a non-expert.

Semiotic heterogeneity Interpreting an entity by humans can cause different interpretations because humans often take the context into account. Considering this problem is not really feasible for a computer because it cannot detect this heterogeneity.

Ordinarily not only one type of heterogeneity appears, rather several ones are involved. Due to all the types and mingled occurrence, it is almost impossible to avoid heterogeneities. Methods are needed to deal with heterogeneity instead of trying to avoid it.
To illustrate varieties of ontology modeling, Figure 1.2 shows two very small ontologies. Both describe parts of a family or more general human beings and their relations among each other. The first ontology, shortened $\mathcal{O}_1$ with namespace 1#, consists of four classes: *1#Person*, *1*#Mother, *1#Father*, *1#Child*. *1#Person* is the superclass of the other three classes. The second ontology, shortened $\mathcal{O}_2$ with namespace 2#, is composed of three classes *2#Human*, *2#Female* and *2#Male* and an object property called *2#hasParent* with domain *2#Human* and complex range *2#Female* $\sqcup$ *2#Male*. By comparing the entity names without any dictionary or other auxiliary information, no straight accordance can be found. With support of

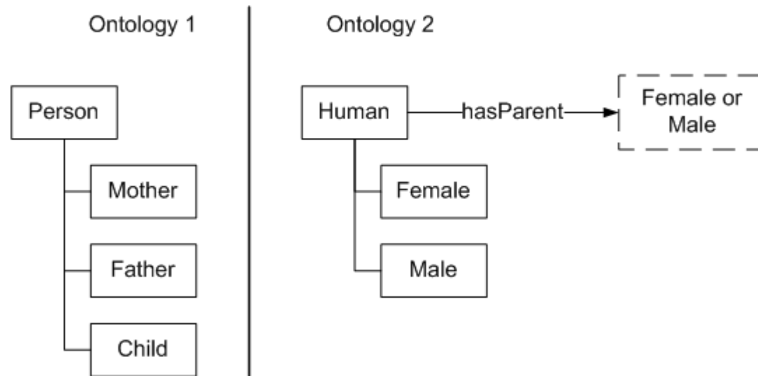

Figure 1.2: Two small ontologies describing humans.

a dictionary it is possible to find *1#Person* $\equiv$ *2#Human*, *1#Mother* and *1#Father* have something to do with *2#hasParent* and also a coherence between *1#Mother* and *2#Female*, *1#Father* and *2#Male*. A structure-level system might detect a connection between *1#Mother*, *1#Father* and *1#Child* as well as *2#Female* and *2#Male* based on the class hierarchy. Some combined methods perhaps recognize a relation between *1#Mother* and *2#Female* and between *1#Father* and *2#Male* because of the subclass structures and *1#Person* being a synonym of *2#Human*. No matcher find a correspondence between *1#Mother* and *2#hasParent* with the special range *2#Female* since complex types are not considered.

Having many different types of unavoidable heterogeneity and detecting that connections can be made up of more than two entities, the need for complex correspondences becomes evident. Therefore clear definitions are required to have a foundation for further work. All following definitions (plain equivalence, plain non-equivalence and complex correspondence) are built upon the definition of correspondence given in Definition 2. Different from Q and R the confidence structure D is not restricted any further.

Definition 4 *A plain equivalence correspondence is a correspondence with* $R = \{=\}$, $Q(O)$ *the set of atomic classes and properties of ontology O.*

Definition 5 *A plain non-equivalence correspondence is a correspondence with* $Q(O)$ *the set of atomic classes and properties of ontology O and no restriction on R.*

Definition 6 *A complex correspondence is a correspondence without any restriction on* R *or* Q*.*

The plain equivalence correspondence is the most frequently used correspondence. Afterwards a few matchers compute plain non-equivalence correspondences but complex ones are hardly ever taken into account. Finding complex correspondences is not really easy but feasible for humans rather than computers. To detect complex types automatically is quite difficult but not impossible at all.

1.3 Related Work

After introducing ontology matching, existing techniques and determining the need for complex correspondences, now related approaches and systems which try to find complex correspondences, give a short review of the current situation.

In order to find complex correspondences some approaches have been considered. For example three out of all thirteen matchers which took part in the OAEI 2008 [4] computed non-equivalence correspondences. Mostly plain non-equivalence correspondences have been regarded, especially subsumptions. A subsumption is a relation between two terms wherein a more general term subsumpts a more specific term. For example an animal is a subsumption of a dog because every dog is also an animal but not every animal has to be a dog.

The three matchers mentioned above identified subsumptions in different ontologies: AROMA [5] in the anatomy ontologies, TaxoMap [9] and Spider [21] in the benchmark ontologies. To get an impression of the systems, their strategies are briefly explained.
TaxoMap creates three types of relationships: equivalence, subclass and semantically relation where equivalence is the similarity of class labels, subclass is adequate to subsumption and semantically stands for all other relations. Finding a subclass relationship is based on the hierarchic class structure. Equivalence relations hold between two classes having linguistic similar labels. All other imaginable links between classes deducible from the set of labels and subclasses are semantically related relationships. Only classes, in particular their labels and subclass relations, are considered and no properties or instances at all. Also correspondences of more than two concepts, complex correspondences, are not taken into account.
AROMA divides the matching process into three parts: pre processing, extraction of rules and post processing. First a set with all relevant data is created for each entity, then subsumptions are computed by using the associated rule model where a rule just connects parts of the sets built in the first step. In the end, a final touch is applied to get better results. Extracting subsumptions based on rules is a different approach that is used by TaxoMap but also only concentrates on subsumptions and

not on complex correspondences.
The last matcher generating plain non-equivalence correspondences is called Spider. Spider not only focuses on subsumptions and wants to bring non-equivalent mappings to the OAEI. To get good results it creates equivalent mappings by using CIDER [8] and non-equivalent by using Scarlet [20]. Selecting and exploring online ontologies helps to find subsumptions, disjoint and named relations. More precisely the relations are obtained by applying derivation rules on the ontologies. The test results for the benchmark ontologies are not very expressive because the OAEI alignments only contain subsumptions and no complex correspondences at all. This resulted in a numerical value less than expected. However, Spider explicit wants to compute complex mappings and to distribute the need of complex correspondences, different from all other matchers mentioned before.

Summarizing all three matching systems computed plain non-equivalence correspondences on their own way but no one regarded complex correspondences just as it has been defined above.

Another approach [23] is based on the inductive logic programming, ILP, and attempts at creating alignments by using the learning theory. Not just this paper is concerned with inductive logic programming and ontology matching, [18] mentioned this topic before. These approaches, different from the three matching systems above, take complex correspondences into account and not only plain or non-plain equivalence correspondences. But here it is not possible to create complex mappings without learning correspondences out of instances. Often ontologies, even the ontologies examined further, do not contain any instances. Hence the learning theory cannot be applied and other strategies have to be adapted in order to find complex correspondences in ontologies without instances.

Altogether by now there exists some matching systems generating non-equivalence mappings and approaches based on the learning theory for complex mappings. Yet if two ontologies without instances are given, complex correspondences cannot be computed. The approach represented in this thesis should rather be like the strategies given by the matching systems but also determine complex correspondences. Similar to TaxoMap, the structure of the entities, not only classes in this case, and their names should be regarded in order to find correspondences. Except for the strategy, the correspondences should also be complex ones like the inductive logic programming approach. Unlike AROMA, finding correspondences is not based on the rule model and also no other ontologies are examined like Spider, especially Scarlet, does. Different from the strategy using the learning theory, instances are not required. As a result there is no existing approach by now which is able to find complex correspondences in the ontologies

examined ongoing.

1.4 Outline and Contribution

Is it possible to detect complex correspondences with state of the art matching techniques? This is the main research question which arised while considering existing matching techniques used by matching systems. Different from other approaches which deal with non-plain correspondences, individuals in the ontologies are not required and this approach is not restricted to correspondences with only atomic entities(plain non-equivalence correspondences). It focuses attention on finding complex correspondences with simple techniques. Therefore the following contributions are accomplished: In the beginning, patterns of complex correspondences in state of the art test sets of the OAEI are detected. After selecting a few ones, algorithms are developed and programmed to automatically find these correspondences. Finally the algorithms are tested on different datasets and the results are analyzed and evaluated.

Introduced the problem statement and ontology matching approaches, in Chapter 2 the way of finding complex correspondences and the classification into several types are described. In the first section, Section 2.1, detecting correspondences and which ontologies have been taken into account is specified, in the second section, Section 2.2, all found types are listed and a short example is given for a better understanding. Additionally a decision is taken which types are implemented later and why. Chapter 3 contains the algorithms of the chosen types and also examples to illustrate correspondences. *Unqualified Restriction*, *Qualified Restriction*, *Value Restriction* and *Property Chain* are the sections and in every section exactly one type is explained. The goal is to provide an impression of the types and how correspondences of such kind can be found. Following Chapter 4 is divided into three sections: Implementation, Settings and Results. In Section 4.1 a general view of the implementation with all external used tools and furthermore the functionality is given. Neither a detailed specification nor a complete analysis is provided because of not setting the focus on the implementation. Section 4.2 contains all preferences of the testing environment which have been declared to be able to reconstruct the testing results in Section 4.3. All results of different tests are indicated and explained in Section 4.3. Finally in Chapter 5 a conclusion is drawn and an overview what has to be done in the future to achieve all introduced ideas is given.

Chapter 2

Exploratory Study

In this chapter the approach of finding complex correspondences is explained further. First the origin of the ideas is indicated and it is described how it is possible to find such correspondences. Afterwards the data on which the examinations occurred are listed. In the second part the types of correspondences which were possible for candidates are explained in detail. Not each candidate was implemented later, just a few one which seemed to be best.

2.1 Approach

First of all complex correspondences had to be found. Some approaches already exist, e.g. the examples presented in [23] and some discussions about correspondences at the OAEI conference track [1]. Existing strategies which are used by some matching systems, like subsumptions, have been also considered. Additionally a few fundamental considerations which occur while building ontologies, e.g. if a class or a property is created or how general range and domain of a property should be, influenced the retrieval of types. In several ontologies the defined types have been searched manually. Another way than considering types and seeking examples in the ontologies was not possible because defining types need first some ideas and a lot of reflection afterwards. At first every ontology was opened in the ontology editor Protege [13] to get an overview of the ontologies and to have a better representation. To find correspondences every entity had to be checked as to whether it is a possible candidate. An automatic way was not possible because no implementation existed so far to find such complex types. The concrete ontologies which were observed: OAEI Conference: SIGKDD, CMT, EKAW, IASTED,

[1] http://keg.vse.cz/oaei/

CONFOF [2] and OAEI Benchmark: 101, 301, 302, 303, 304 [3] which are all written in the ontology language OWL. In every OAEI Conference ontology the domain conference is specified by classes and properties. Describing the domain conference seems to be suitable [24] because most persons dealing with ontologies are academics and know this topic already. Therefore it is easier to understand the complex correspondences mentioned further instead of examples in an unfamiliar domain. The OAEI Benchmark ontologies attend the domain bibliography which is also well-known by academics. Another reason for choosing these ontologies are the existing and available alignments. For the conference set every combination of ontologies is possible because for every pair of two ontologies an alignment is available and such an alignment is used during the matching process. Only for each combination with the ontology 101 an alignment was available for the benchmark ontologies. Why an alignment with plain equivalence correspondences is required, is described in Chapter 3. At the moment it is only useful to bear in mind that not every combination of two ontologies has been regarded while browsing through the ontologies. Otherwise some following explanations might not be comprehensible.

2.2 Types of Complex Correspondences

After determining the complex types they have to be written down formally and explained more precisely. For that reason examples are shown and explained. In the end a decision which types are implemented later is taken based on the occurrences and some other criteria.

In Figure 2.1 two different ontologies are illustrated. Both describe a conference because this domain seems to be reasonable for an academic reader. Rectangles represent classes, ellipses datatypes like string or date and arrows stand for properties linking a class to another class (object property) or to a datatype (datatype property).

Every type found in the exploration is listed below with a small example out of Figure 2.1. Also a description with information about the reason for having differences is given to understand why a certain type may occur. Additionally imaginable matching results of conventional matchers are given to get an impression of their operating principles and findings. Whenever similarity between entities is mentioned, a string similarity is meant although it is not specified any further. For formal definitions assume two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, class $\mathcal{C}_1$, object property $\mathcal{OP}_1$, datatype property $\mathcal{DP}_1$ in $\mathcal{O}_1$ and class $\mathcal{C}_2$, object property $\mathcal{OP}_2$ and datatype

[2] http://nb.vse.cz/ svabo/oaei2008/

[3] http://oaei.ontologymatching.org/2008/benchmarks/

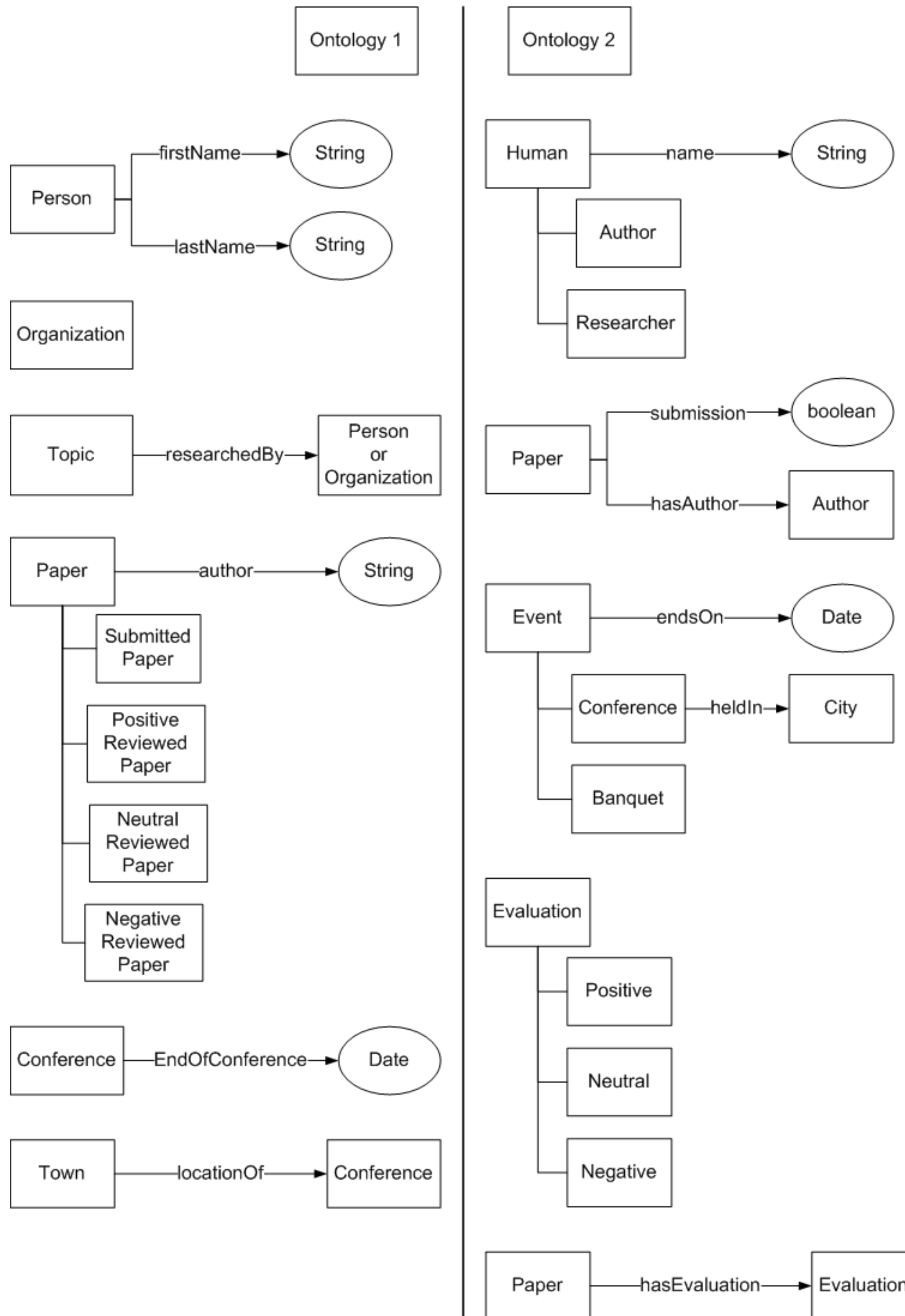

Figure 2.1: Two example ontologies for illustrating the types of complex correspondences

properties $\mathcal{DP}_2$, $\mathcal{DP}_2$' in $\mathcal{O}_2$. All correspondences are in first-order-logic because they can be described in a simple way and first-order-logic is well-known not only in a specific field. Description logic was not selected because of not being able to express all complex types.

An equivalence between both parts of the correspondence was chosen although there are some existential quantifiers, e.g.
$\forall$x (*2#Researcher*(x) $\leftrightarrow$ $\exists$y (*1#researchedBy*(y,x)) $\wedge$ *1#Person(x))*
and this might be confusing. The implication from the complex term to the simple term is the easy one and claims if anything exists which is researched by a person the person has to be a researcher. On the other hand, a researcher in this case is a person and has done some research otherwise one would not be a researcher. As soon as the left side is available, the variables on the right side are defined automatically.

Subsumption

Explanation A *subsumption* is a generalization of one term to another and can arise if in one ontology a more general concept is needed that is different from another ontology. They can be found by using dictionaries/thesauri or sometimes by comparing substrings. It is not a complex correspondence, but a plain non-equivalence one.

Formally $\forall$x ($\mathcal{C}_2$(x) $\rightarrow$ $\mathcal{C}_1$(x))
$\mathcal{C}_1$ is a *subsumption* of $\mathcal{C}_2$ or $\mathcal{C}_1$ is a generalization of $\mathcal{C}_2$.

Example $\forall$x (*2#Author(x)* $\rightarrow$ *1#Person(x)*)
Every author is also a person but not every person is necessarily an author. If an equivalence between *1#Person* and *2#Human* exists, the *subsumption* found above is trivial because of deducing a subclass relation between *1#Person* and *2#Author*.

Matcher results Some matchers already compute *subsumptions* but most ones do not find the correspondence or make a plain equivalent correspondence out of it instead of a *subsumption*.

Property Chain

Explanation While creating an ontology a creator must decide whether to model an entity as a class or as a property. Instead of a property which relates two classes or a class and a datatype, one can also construct a new class and connect it to other classes.

Formally $\forall$x ($\mathcal{DP}_1$(x,datatype) $\leftrightarrow$ $\exists$y ($\mathcal{OP}_2$(x,y) $\wedge$ $\mathcal{DP}_2$(y,datatype)))
In the end both expressions relate a class to a datatype and describe the same issue but in a different way.

Example $\forall$x (*1#author(x,string)* $\leftrightarrow$ $\exists$y (*2#hasAuthor(x,y)* $\wedge$ *2#name(y,string)*))
The datatype property *1#author* maps a *1#Paper* to the name of its author as a string. Instead *2#hasAuthor* relates a *2#Paper* to a *2#Author*. Every *2#Author* has a name given by the datatype property *2#name*. If a *2#Author* exists which is the author of a *2#Paper* and has a name, it is equivalent to the property *1#author*.

Matcher results Conventional matching systems only concentrate on correspondences between two entities. Such a matcher perhaps finds similarities between the properties but not a correspondence between one property and a composition of two properties.

Range Restriction Often complex ranges are used and just a subclass of the range might lead to a correspondence. There are several different types which are described further. All these types are not supported by any matcher. Every following type is based on the term $\forall$x (*1#A(x)* $\leftrightarrow$ *2#B(x)* $\exists$y (*2#R(x,y)* $\wedge$ *2#C(y)*)) where *A*,*B*,*C* are classes and *R* is a property. Not every class of the second ontology has to be restricted but at least one of them per type.

Unqualified Restriction

Explanation Classes can often also be modeled as object properties. Where to create a class or a property is in the eye of the builder and cannot be standardized. There can also be differences in ontologies if the names of classes or properties are the same but have another meaning, e.g. a seller can be a person or a whole company. If a class $\mathcal{C}_1$ in $\mathcal{O}_1$ describes a seller as a person and an object property $\mathcal{OP}_2$ in $\mathcal{O}_2$ explains that someone bought something from a seller, the seller can be a person or a company. Correspondences of type *Unqualified Restriction* exactly deal with this problem, where a class is namely similar to an object property with a complex range. Additionally the class is namely similar to the object property. To get this type out of the basic term described above, the variables x and y are changed in the property to get *R(y,x)* instead of *R(x,y)* and *B(y)* is left off. This type is called *Unqualified Restriction* because the restricted range is not qualified further by an existential quantifier.

Formally $\forall$x ($\mathcal{C}_1$(x) $\leftrightarrow$ $\exists$y ($\mathcal{OP}_2$(y,x) $\wedge$ $\mathcal{R}$'(x)))
where $\mathcal{R}$' is a subclass of the range of $\mathcal{OP}_2$.

Example $\forall$x (*2#Researcher(x)* $\leftrightarrow$ $\exists$y (*1#researchedBy(y,x)* $\wedge$ *1#Person(x)*))
In $\mathcal{O}_2$ a *2#Topic* can be *2#researchedBy* a *2#Person* or a whole

2#Company. $\mathcal{O}_1$ contains a class called *2#Researcher* which is equivalent to the construct out of $\mathcal{O}_2$ if a *2#Topic* exists.

Matcher results Similarity between *2#Researcher* and the property *1#researchedBy* or maybe a *subsumption* from *1#Person* to *2#Researcher* might be found by an existing matcher.

Qualified Restriction

Explanation This type is very similar to *Unqualified Restriction* but instead of holding a string similarity between a class and an object property, the similarity here is between the class and the range of an object property. To get this type out of the basic term only *B(x)* has to be deleted. The name *Qualified Restriction* is chosen because of qualifying the restricted range additionally by an existential quantifier different from *Unqualified Restriction*.

Formally $\forall$x ($\mathcal{C}_1$(x) $\leftrightarrow$ $\exists$y ($\mathcal{OP}_2$(x,y) $\wedge$ $\mathcal{R}$'(y)))
where $\mathcal{R}$' is a subclass of the range of $\mathcal{OP}_2$. In distinction from *Unqualified Restriction* where in the term $\mathcal{R}$'(x) the variable x is qualified by an universal quantifier in $\mathcal{R}$'(y) the variable y is qualified by an existential quantifier.

Example $\forall$x (*1#PositiveRewiedPaper(x)* $\leftrightarrow$ $\exists$y (*2#hasEvaluation(x,y)* $\wedge$ *2#Positive(y)*))
A *1#PostiveReviewedPaper* is equivalent to a *2#Paper* which has a positive Evaluation. Range of *2#hasEvaluation* is *2#Evaluation* and not just *2#Positive*. Without determining this fact, a correct correspondence cannot be found.

Matcher results Similarity between the class *1#PositiveRewiedPaper* and the subclass of the range *2#Positive*.

Value Restriction

Explanation Sometimes a class can be expressed as a datatype property with a finite datatype, e.g. boolean, or a set of values as range. This type is a special case of *Qualified Restriction* with a datatype property instead of a object property. It is called *Value Restriction* because of restricting a datatype range on a particular value.

Formally $\forall$x ($\mathcal{C}_1$(x) $\leftrightarrow$ $\exists$y ($\mathcal{DP}_2$(x,y) $\wedge$ y=value))
The name of the class and the name of the property are mostly similar, especially if the value should declare that a certain case is present. If a datatype range is not finite one normally cannot draw any conclusion because of not knowing which value has a special meaning.

Example $\forall$x (*1#submittedPaper(x)* $\leftrightarrow$ $\exists$y (*2#submission(x,y)* $\wedge$ y=true))
A *1#submittedPaper* is equivalent to *2#submission* of a *2#Paper* if the datatype value is true.

Matcher results Existing matching systems may find a similarity between *1#submittedPaper* and *2#submission* but this is not a correct correspondence because of comparing a class to a property which is not indeed sensible in this case.

Domain Restriction

Explanation Beside ranges domains also can be restricted.

Formally $\forall$x,y ($\mathcal{DP}_1$(x,y) $\leftrightarrow$ $\mathcal{DP}_2$(x,y) $\wedge$ $\mathcal{D}$'(x)) or
$\forall$x,y ($\mathcal{OP}_1$(x,y) $\leftrightarrow$ $\mathcal{OP}_2$(x,y) $\wedge$ $\mathcal{D}$'(x)) or
$\forall$x ($\mathcal{C}_1$(x) $\leftrightarrow$ $\forall$y ($\mathcal{OP}_2$(y,x) $\wedge$ $\mathcal{D}$'(y)))
where $\mathcal{D}$' is a subclass of the domain. Only if the domain is restricted the correspondence holds.

Example $\forall$x,y (*1#EndOfConference(x,y)* $\leftrightarrow$ *2#endsOn(x,y)* $\wedge$ *2#Conference(x)*)
The *1#endOfConference* is the same as *2#Conference 2#endsOn* a special date. The *2#endsOn* property has the domain *2#Event* which is a superclass of 2#*Conference*. Without restricting the domain a correspondence is not given.

Matcher results A matcher might find a correspondence between both properties/class and range but without considering the domains.

Compound Property

Explanation Some properties can be divided into two properties which are compounded similarly to the result of one property. For example a telephone number can be divided into the dialing code and the number. Thus one telephone number property or two properties from the dialing code and number can be created.

Formally $\forall$x,y ($\mathcal{DP}_1$(x,y) $\leftrightarrow$ $\exists$z,z' ($\mathcal{DP}_2$ (x,z) $\wedge$ $\mathcal{DP}_2$'(x,z') $\wedge$ $f(z, z') = y$))
where f is a in general an arbitrary function but in this case f concatenates z and z'. For example $f("a", "b") = "ab"$ is the concatenation of $"a"$ and $"b"$.

Example $\forall$x,y (*2#name(x,y)* $\leftrightarrow$ $\exists$z,z' (*1#firstName(x,z)* $\wedge$ *1#lastName(x,z')* $\wedge$ f(*1#firstName(x,z)*, *1#lastName(x,z')*) = *2#name(x,y)*))
The name of a person normally consists of the first name and the last name. The first and last name of a human is compounded the same as a name of a person because person is a synonym for human.

Matcher results If only mapping one-to-one a matcher may find two correspondences or maybe *subsumptions* but not a connection between three properties.

Inverse Property

Explanation Many properties can be reversed by changing domain and range. There are some key words like *by* or *of* which can indicate this type.

Formally $\forall$x,y ($\mathcal{OP}_1$(x,y) $\leftrightarrow$ $\mathcal{OP}_2$(y,x))
If domain and range of one property are reversed, a correspondence can be found.

Example $\forall$x,y (*2#heldIn(x,y)* $\leftrightarrow$ *1#locationOf(y,x)*)
2#heldIn has the domain *2#Conference* and the range *2#City*, *1#locationOf* the domain *1#Town* and the range *1#Conference*. *1#Town* and *2#City* can be seen as synonyms and reversing domain and range of one property generates a correspondence.

Matcher results Matcher analyzing the structure usually recognize a difference between ranges and domains. Also element-level matchers cannot find a correspondence since the names are not always similar.

After identifying and discovering the correspondences some types have to be chosen. Implementing every type was not possible due to time restriction. To come to a decision Table 2.1 with the numbers of the found correspondences per type has been created. The displayed numbers are lower bounds because there might be correspondences which have not been found. Not every complex type has been searched in the benchmark ontologies only those types where no example was found in the conference set. Therefore Table 2.1 contains "-" whenever a type of complex correspondence has not been considered in these ontologies. Having this first overview advantages and disadvantages of every type have to be considered. Then every type is listed with an explanation whether it has been taken or not.

Subsumption Some matchers already compute *subsumptions* and it is quite easy to find them with a dictionary. The implementation is also not very different if a lexical database like Wordnet [7] is integrated because the system can return all generalizations of one word. Moreover it is only a plain non-equivalence type and not a complex one. Although most correspondences are of this type, it was not attractive enough to implement and therefore not selected.

Type	Benchmark	Conference	Total
Subsumption	-	13	13
Property chain	9	0	9
Domain restriction	-	4	4
Compound property	4	0	4
Inverse property	-	5	5
Value Restriction	-	2	2
Unqualified Restriction	-	1	1
Qualified Restriction	-	2	2
Range restriction total	-	5	5

Table 2.1: Number of found correspondences per type

Property chain Correspondences of type *Property Chain* are quite often available in the benchmark ontologies. It is understandable that it is possible to create one property or two properties expressing the same in the end. Not only limited to some ontologies this type of correspondence is conceivable. Implementing the type requires to create an algorithm which is universally usable for every pair of ontologies. Beside name comparisons of particular entities, which belongs to the element-level, structure-level parts must also be integrated to find this correspondences. Due to the unrestrictedness on special ontologies and the need for different techniques at the computation, this type of correspondences was selected for implementation.

Range restriction Not every property is only used in one way, with a complex range it can be used in different ways. Restricting the range of a property can sometimes lead to an equivalence to another property or a class. Altogether the range restrictions can be found a few times in the given ontologies.

Unqualified Restriction Having a property with complex range and a class similar to the property can lead to a correspondence called *Unqualified Restriction*. The property can be used to express different issues and only if the range is restricted to one special class is a correspondence between the class and the property with its range given. Modeling an entity as a property or as a class is not predetermined and everybody can choose which version to use. Also if only one example has been found in the ontologies it is a fundamental distinction and has been picked out to be implemented.

Qualified Restriction Here the type also relates to the problem where to

create a class and where to pick a property. It is quite similar to the type *Unqualified Restriction* but not similar enough to use the same algorithm. Two examples are given in the ontologies but there might be more in other ontologies. Because of its advantages, especially the generality of this type not restricted on some ontologies rather having differences in modeling and the number of occurrence in the examined ontologies, this type was also selected.

Value Restriction A range restriction based on a datatype property and a special value of its range. In general one can create a class or explain the same circumstance with a datatype property and fixed value. Only two examples have been found in the ontologies but the correspondence seems to describe a general difference in modeling which occurs as often as other types but can be important. Due to the observations above this type appears to be a good choice.

Implementing this type, especially the three subtypes, requires element-level and also structure-level techniques which require some deliberations and the quantity of strategies is less limited. The type *Range Restriction* has been selected because of being quite general and promising.

Domain restriction Some properties/classes have a more general meaning than other ones even if their names in the ontologies are the same. If the ranges of two properties are equal they might differ in their domain. Also a class can be equivalent to a property with a special domain. In all examined ontologies there have only been a few examples and the restriction on the range which is contrary to the *Domain Restriction* seemed to be more promising and interesting. Especially no need for a distinction between datatype and object property limits the variety of correspondences. Because of implementing the *Range Restriction* type which is similar to the *Domain Restriction* type it has not been chosen.

Compound property Datatype properties, especially with range string can sometimes be divided into more than one datatype property. The compound values are at the end the same as the value of one property. There are several examples but they did not seem to be very universal because of concentrating on persons with their name and address. Other cases are difficult to imagine because other strings can be rarely separated into substrings which really make sense. Due to this fact the type *Compound Property* was not selected for implementation.

Inverse property In an ontology a property generally can connect one class to

another or vice versa because a property is not a symmetric relation. A consistent way of modeling a property with its domain and range does not exist. Finding inverse properties is not just difficult because if two properties are namely similar and maybe one property contains a key word like *by* or *of*, only a comparison between exchanged domain and range of one property and domain and range of the other property is enough to determine a correspondence. Additionally in the examined ontologies often one ontology contains the ordinary property and also its inverse. In such a case finding a correspondence between the inverse property and another property can already be derived from the plain equivalence correspondences. Besides it is also only a plain non-equivalence type like *subsumption*. Although some examples have been found, this type was not as interesting as other types because the algorithm is not very difficult and many correspondences may be trivial.

To sum up the choice of the types, *Property Chain* and *Range Restriction* are further implemented because of assuming the best success, having a few expressive examples, being very general and not restricted to only some cases or ontologies. Implementing four algorithms also appears appropriate for this thesis.

Chapter 3

Algorithms

Introduced ontology matching, existing strategies and some problems, the need for complex alignments became comprehensible. After finding some complex correspondences, several types were presented and a few of them selected. In this chapter the implemented algorithms of the chosen types are explained. Furthermore a detailed example to every algorithm illustrates the theoretical description. All algorithms are implemented and the testing results can be found in Chapter 4.

Development and implementation of the algorithms are based on the correspondences found in all ontologies. The goal was to write an algorithm which detects at least these examples but should not contain too ridic constraints so that also other correspondences can be discovered. Additionally the algorithms should be as simple as possible and not unnecessarily difficult. Every algorithm includes several loops to iterate through elements and conditional statements to select the correct entities. After identifying all essential components, the order of the loops and statements had to be defined. Changing the order may influence intermediate results and also the runtime. To get the best performance an order has been chosen which achieves the least runtime.

Moreover the algorithms have two ontologies and one alignment as input. Building a bridge between the two ontologies is achieved by the conventional alignment and with its help it is for example possible to get all superclasses of a class out of both ontologies and not only out of the ontology in which the class is contained. A more precise explanation of the functionality is given further when describing the logical notions.

For the sake of simplicity only the names of the entities and namespaces like 1#

and 2# are used instead of whole URIs with the complete namespace. Shortened also the abbreviations $\mathcal{O}_1$, $\mathcal{O}_2$ are defined rather than Ontology 1 and Ontology 2. Also some other abbreviations are used: $\mathcal{C}$ for class, $\mathcal{P}$ for property and $\mathcal{E}$ for entity. Certain notions are used to compute the correspondences:

logical notions The logical notions are defined in the ontology language OWL in which all considered ontologies are expressed. When defining an ontology these concepts were already mentioned.

subClassOf The subClassOf relation is adequate with the specialization where one class is more specific than another. It is the counterpart of superClassOf which is described later on. In the algorithms a method called subClassOf($\mathcal{C}$, $\mathcal{O}$) is used to compute the subClasses of a class $\mathcal{C}$ in ontology $\mathcal{O}$.

superClassOf If one class is a generalization of another class. Two methods are used in the algorithms based on the superclass concept. First one is superClassOf($\mathcal{C}$, $\mathcal{O}$) and returns all classes of $\mathcal{O}$ which are a superclass of $\mathcal{C}$. The second on is isInAlignedSuperclasses($\mathcal{C}$, $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$) which additionally considers the reference alignment $\mathcal{A}$ to find all superclasses of $\mathcal{C}$ not only of $\mathcal{O}_1$ but also of $\mathcal{O}_2$. For a better understanding, in Figure 3.1 an example is illustrated. On the left side are the classes of $\mathcal{O}_1$, on the right the classes of $\mathcal{O}_2$. In the middle the reference alignment is located. A plain equivalent correspondence between *1#*$\mathcal{C}_3$ and *2#*$\mathcal{C}_1$ can be found in the alignment. Imagine all superclasses of *2#*$\mathcal{C}_1$ in both ontologies are requested. Only considering $\mathcal{O}_2$ do not deliver any results because in $\mathcal{O}_2$ exists no superclass of *2#*$\mathcal{C}_1$, only the subclasses *2#*$\mathcal{C}_2$ and *2#*$\mathcal{C}_3$. Having the correspondence makes it possible to check the superclasses in $\mathcal{O}_2$ by regarding all superclasses of *1#*$\mathcal{C}_3$. In this case *1#*$\mathcal{C}_1$ has a superclass called *1#*$\mathcal{C}_1$. Now *1#*$\mathcal{C}_1$ can be declared as a superclass of *2#*$\mathcal{C}_1$ although they are not in the same ontology. Determining subclasses out of both ontologies is of course analog because of being the counterpart of superclasses.

domain Domain of a property is a relation to indicate which classes are permitted to be part of this property as domain. In the algorithms the method domain($\mathcal{P}$, $\mathcal{O}$) is used to get the domain of a property $\mathcal{P}$ which is contained in ontology $\mathcal{O}$. Beside the classes defined as domain, also every subclass is returned. For example if in ontology $\mathcal{O}$ a property $\mathcal{P}$ has the defined domain $\mathcal{C}_1$, whereby $\mathcal{C}_1$ has the subclasses $\mathcal{C}_2$ and $\mathcal{C}_3$.

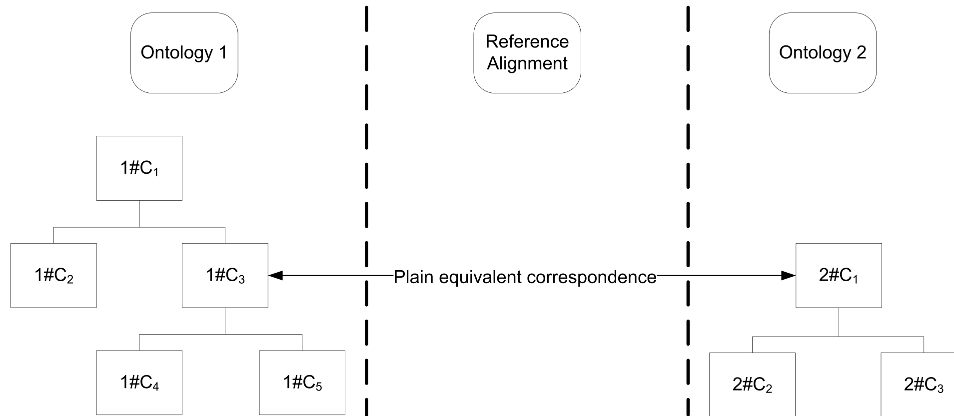

Figure 3.1: Finding subclasses in two ontologies with the help of a reference alignment

Now if domain($\mathcal{P}$, $\mathcal{O}$) is called, this method returns $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ as the domain of $\mathcal{P}$.

range Range is the opposite of domain and defines which classes can be in the range of a property if an object property is available or which datatypes in the case of a datatype property. Calling range($\mathcal{P}$, $\mathcal{O}$) returns on one hand all classes of $\mathcal{O}$ defined as range and their subclasses as with the domain if $\mathcal{P}$ is an object property and on the other hand the defined datatype in $\mathcal{P}$ is a datatype property.

disjoint Two classes are disjoint if their intersection is empty. In the algorithms it is indicated by $\mathcal{C}_1 \sqsubseteq \neg\, \mathcal{C}_2$ if $\mathcal{C}_1$ is disjoint with $\mathcal{C}_2$.

equivalent Having a class $\mathcal{C}_1$ equivalent to another class $\mathcal{C}_2$ means that $\mathcal{C}_1$ is subclass of $\mathcal{C}_2$ and simultaneously $\mathcal{C}_2$ is subclass of $\mathcal{C}_1$. In general one could say if two classes are equivalent, they claim the same issue otherwise they are not. Equivalence is not immediately used in the algorithms, but its counterpart. Having two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ the non-equivalence is indicated by $\mathcal{C}_1 \,! = \mathcal{C}_2$.

syntactical notions Beside the logical notions, several other characteristics, not defined in OWL, are needed in order to find complex correspondences.

similar Similar entities have a high similarity value measured against some similarity criteria. For computing these values a conventional matcher

can be used as long as it is mostly based on a linguistic approach, like [11] or [10]. In this case a matcher computing the values based on the Levenshtein measure [15], which computes how many operations (insert, delete, modify) are needed to get one word out of the other, has been applied. Having values for every pair of entities, a decision which entities are similar has to be taken. Therefore a particular variable for every similar relation acts as threshold and if a value is greater than this threshold, the corresponding entities are selected as being similar. A method isSimilar($\mathcal{E}_1$, $\mathcal{E}_2$) is used to determine the similarity value of $\mathcal{E}_1$ and $\mathcal{E}_2$. It returns true if the value is higher than a defined threshold and false otherwise. The thresholds do not appear in the algorithms because they can be modified.

head noun A name can be separated into two parts: head noun and not head noun. The head noun is the main word of a name and can be find by applying some rules. If the name only contains one word, the whole name is the head noun. Having a name compound of at least two words, some keywords exist, for example *of* or *by*, which indicates that the head noun is directly before the keyword, e.g. author of paper where author is the head noun. Otherwise if the names consists of more than one word and additionally no keyword is contained, the last word of the name is most often the head noun, e.g. paper author where author is the head noun. Now after defining the set of rules, the head noun of a name can be determined easily. Calling the method nameWithoutHeadNoun($\mathcal{E}$) returns the name of the entity $\mathcal{E}$ without containing the head noun.

first part Aside from dividing a name into head noun and not head noun, a name can also be partitioned into first part and other part. First part is just the first word of a name and other part are all remaining words except for the first one. If the name contains only of one word, the other part is empty. In the algorithms the method firstPartOfName($\mathcal{E}$) returns a string with the first word of the name of the given $\mathcal{E}$.

reduced name A reduced name is a name where a certain part is deleted in contrast to the whole name. In this case the word of the name is deleted which has the highest similarity value with a name of a class in the other ontology. Reducing a name is implemented by the method reducedName($\mathcal{E}$, $\mathcal{O}$) and returns the reduced name of $\mathcal{E}$ where $\mathcal{E}$ belongs to ontology $\mathcal{O}$.

compatible Two datatypes are compatible if one datatype can be translated into the other and vice versa. String is compatible to every other

datatype but for example date is not compatible to boolean. The method isCompatible($\mathcal{R}_1$, $\mathcal{R}_2$) checks if the range $\mathcal{R}_1$ from a datatype property is compatible to a range $\mathcal{R}_2$ from another datatype property.

Beside the methods mentioned above, a few more are used in the algorithms: classes($\mathcal{O}$), objectProperties($\mathcal{O}$) and datatypeProperties($\mathcal{O}$). They are used to deliver data and just return the corresponding set of entities from an ontology $\mathcal{O}$. Having described all notions and required methods, the algorithms can be explained afterwards.

Beside the algorithms, examples which can be found in the ontologies should help to understand the technical description. All four Figures 3.2, 3.3, 3.4, 3.5 consist of classes, recognized by rectangles and properties recognized by ellipsis. Characteristics of entities like *range*, *similar*, *not equal* and so on, are represented by labeled lines or labeled arrows if the direction is important. Moreover datatypes are displayed as hashes to indicate a datatype property. Additionally all other appearing parts like head noun/not head noun or any other division of a name into several words are indicated by hexagons. Every part of a correspondence is bold framed to quickly detect the relevant concepts. For reasons of simplification all names of entities are noted without their namespace, but they are unique anyway.

3.1 Unqualified Restriction

Unqualified Restriction describes a complex correspondence between a class and an object property with restricted range. A characteristic feature to distinguish this type from especially the other range restrictions is the similarity of class and property. The following explanations refer to Algorithm 1.

First a class $1\#c$ of $\mathcal{O}_1$ and an object property $2\#p$ of $\mathcal{O}_2$ must be given to check over if all other requirements for a correspondence of type *Unqualified Restriction* are fulfilled. Therefore the algorithm iterates through all classes of $\mathcal{O}_1$ and all object properties of $\mathcal{O}_2$. Further the range $2\#r$ and domain $2\#d$ of $2\#p$ are determined. Also the name of $1\#c$ has to be similar to the name of $2\#p$. On the basis of the similarity, entities describing the same issue should be identified. A threshold, which can be modified, defines how similar the names must be to get a satisfaction. Moreover every superclass $2\#s$ of $1\#c$ is regarded. This is only possible if an alignment of $\mathcal{O}_1$ and $\mathcal{O}_2$ is available, because otherwise only the superclasses of $\mathcal{O}_1$ can be determined. The last condition is the conjunction of several requirements. Only if all three statements are true, can a correspondence be found. First $2\#s$ has to be a subclass of $2\#r$ to ensure that $2\#p$ maps $2\#d$ onto a more general concept than $1\#c$. Otherwise the correspondence is not very meaningful because $1\#c$ and $2\#p$ with restricted range $2\#s$ should describe the same issue.

Excepting the identity of $1\#c$ and $2\#s$ avoid unnecessary correspondences, e.g. finding *Organizer(y) = organizedBy(x,y) Organizator(y)* is of little avail if *Organizator* has the same meaning in both ontologies. Having the domain $2\#d$ being disjoint with $2\#s$ guarantees the relation between two widely different classes represented by $2\#p$. Else domain and range could be the same or connect related classes which is unintended. If all these conditions are fulfilled a correspondence between $1\#c$, $2\#p$ and $2\#s$ is found. In first-order-logic the correspondence looks like $1\#c(x) \leftrightarrow \exists y\ (2\#p(y,x) \wedge 2\#s))$.

To illustrate the algorithm an example is shown in Figure 3.2. *Organizator* on the left side is the class out of $\mathcal{O}_1$ which is similar to the object property *organised_by* of $\mathcal{O}_2$. Superclass of *Organizator* is *Person*, this is figured out by regarding the alignment. Further the defined range of *organised_by* is *Organisation* ∪ *Person*. For a union of classes every included class is also a subclass of the union. That is why *Person* is in the set of ranges and also the superclass as already mentioned. Continuing the algorithm, class and range, here *Organizator* and *Person* are not equivalent. To reach a complex correspondence of type Unqualified Restriction domain of the property and the superclass must be disjoint. In the figure one can see the disjointness of *Event*, which is the domain of *organised_by*, with *Organizator*. All requirements are now satisfied and a correspondence is found. Formally the correspondence is: ∀x (*Organizator(x)* ↔ ∃y (*organised_by(y,x)* ∧ *Person(x)*)).

Algorithm 1 Find Unqualified Restriction correspondences

UNQUALIFIED RESTRICTION($\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$)

1: **for** $1\#c$ in CLASSES($\mathcal{O}_1$) **do**
2:    **for** $2\#p$ in OBJECTPROPERTIES($\mathcal{O}_2$) **do**
3:       $2\#r \leftarrow$ RANGE($2\#p$, $\mathcal{O}_2$)
4:       $2\#d \leftarrow$ DOMAIN($2\#p$, $\mathcal{O}_2$)
5:       **if** ISSIMILAR($1\#c$, $2\#p$) **then**
6:          **for** $2\#s$ in ALIGNEDSUPERCLASSES($1\#c$, $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$) **do**
7:             **if** $2\#s \sqsubseteq 2\#r$ **and** $1\#c$!= $2\#r$ **and** $2\#d \sqsubseteq \neg 2\#s$ **then**
8:                ▷ *create* ∀*x* (*1#c(x)* ↔ ∃*y* (*2#p(y,x)* ∧ *2#s*))
9:             **end if**
10:          **end for**
11:       **end if**
12:    **end for**
13: **end for**

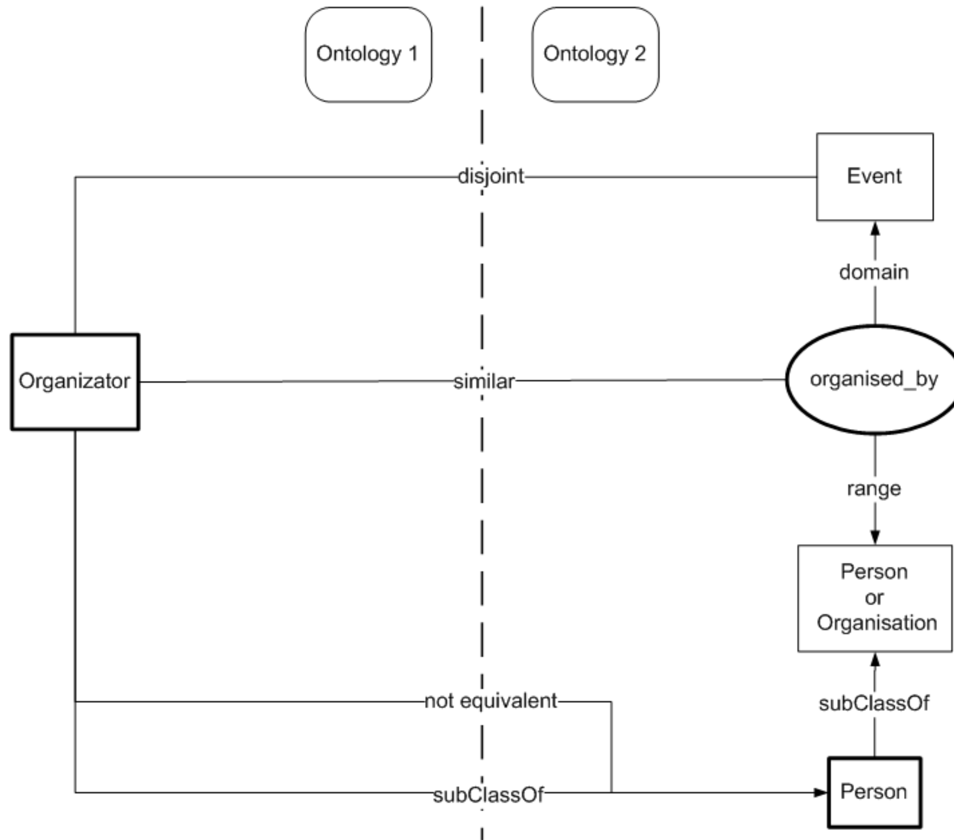

Figure 3.2: Correspondence of type Unqualified Restriction between *Organizator* and *organised_by* with restricted range *Person*. The class *Organizator* can be found in the OAEI conference ontology sigkdd, *organised_by* and its range in the OAEI conference ekaw.

3.2 Qualified Restriction

A class, an object property and additionally one part of the class name being similar to the restricted range, indicates a complex correspondence called *Qualified Restriction*. Algorithm 2 describes the process of finding correspondences of this type. First the classes of $\mathcal{O}_1$ and $\mathcal{O}_2$ are regarded. Having two fixed classes $1\#c$ and $2\#c$ the object properties of $\mathcal{O}_1$ are examined further. $1\#r$ is assigned to the range of $1\#p$, $1\#d$ to the domain of $1\#p$. Only if the name without head noun of $1\#c$ and $2\#c$ are similar, does the algorithm continue with these classes. Afterwards the superclasses of $1\#c$ are computed, this time just the superclasses of $\mathcal{O}_1$ because here the other ones are not needed. If the current superclass $1\#s$ is similar to the name of $1\#p$ another necessary condition is fulfilled. Moreover a superclass $1\#s'$ of $2\#c$ is required and can be charged by considering the alignment. Now the entities have to satisfy three more conditions. $1\#s$ has to be a subclass of $1\#r$, this guarantees that $1\#c$ can be applied as range. Additionally both classes $1\#c$ and $2\#c$ should not be the same to avoid impractical correspondences. Further $1\#s'$ has to be a subclass of $1\#d$ to be able to deploy $2\#c$ as range for $1\#p$. Once all these requirements are fullfilled, a correspondence of type *Qualified Restriction* has been found and can be expressed as following: $\forall$x (2#c(x) $\leftrightarrow$ $\exists$y (1#p(x,y) $\wedge$ 1#c(y))).

Figure 3.3 shows an example of a *Qualified Restriction*. On the left side the class *Accepted_Paper* can be split up into Accepted and Paper, where Paper is the head noun of the expression and Accepted not the head noun. Accepted is similar to Acceptance, but *Accepted_Paper* is not equivalent to *Acceptance*. The object property *hasDecision* accomplishes all conditions required by the algorithm: *Acceptance* has a superclass called *Decision* which is also the range of *hasDecision*, additionally Decision and hasDecision are similar. Moreover the domain of *hasDecision* is *Paper* and *Paper* on the other hand is a superclass of *Accepted_Paper* because in the ontologies *Accepted_Paper* is a subclass of *Paper* in its own ontology and *Paper* $\in \mathcal{O}_1 \equiv$ *Paper* $\in \mathcal{O}_2$ given in the alignment. In the end a correspondence between *Accepted_Paper* and *hasDecision* with *Acceptance* is found. Formally it can be expressed that way: $\forall$x (*Accepted_Paper(x)* $\leftrightarrow$ $\exists$y (*hasDecision(x,y)* $\wedge$ *Acceptance(y)*)).

3.3 Value Restriction

The existence of a datatype property with a finite range is characteristic for this type of complex correspondence. Finding such correspondences is described in Algo-

Algorithm 2 Find Qualified Restriction correspondences

QUALIFIED RESTRICTION($\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$)

1: **for** $1\#c$ in CLASSES($\mathcal{O}_1$) **do**
2:   **for** $2\#c$ in CLASSES($\mathcal{O}_2$) **do**
3:     **for** $1\#p$ in OBJECTPROPERTIES($\mathcal{O}_1$) **do**
4:       $1\#r \leftarrow$ RANGE($1\#p$, $\mathcal{O}_1$)
5:       $1\#d \leftarrow$ DOMAIN($1\#p$, $\mathcal{O}_1$)
6:       **if** ISSIMILAR(NAMEWITHOUTHEADNOUN($1\#c$), $2\#c$) **then**
7:         **for** $1\#s$ in SUPERCLASSES($1\#c$, $\mathcal{O}_1$) **do**
8:           **if** ISSIMILAR($1\#s$, $1\#p$) **then**
9:             **for** $1\#s'$ in ALIGNEDSUPERCLASSES($2\#c$, $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$) **do**
10:               **if** $1\#s \sqsubseteq 1\#r$ **and** $1\#c$!= $2\#c$ **and** $2\#s' \sqsubseteq 1\#d$ **then**
11:                 $\triangleright$ *create* $\forall x\ (2\#c(x) \leftrightarrow \exists y\ (1\#p(x,y) \wedge 1\#c(y)))$
12:               **end if**
13:             **end for**
14:           **end if**
15:         **end for**
16:       **end if**
17:     **end for**
18:   **end for**
19: **end for**

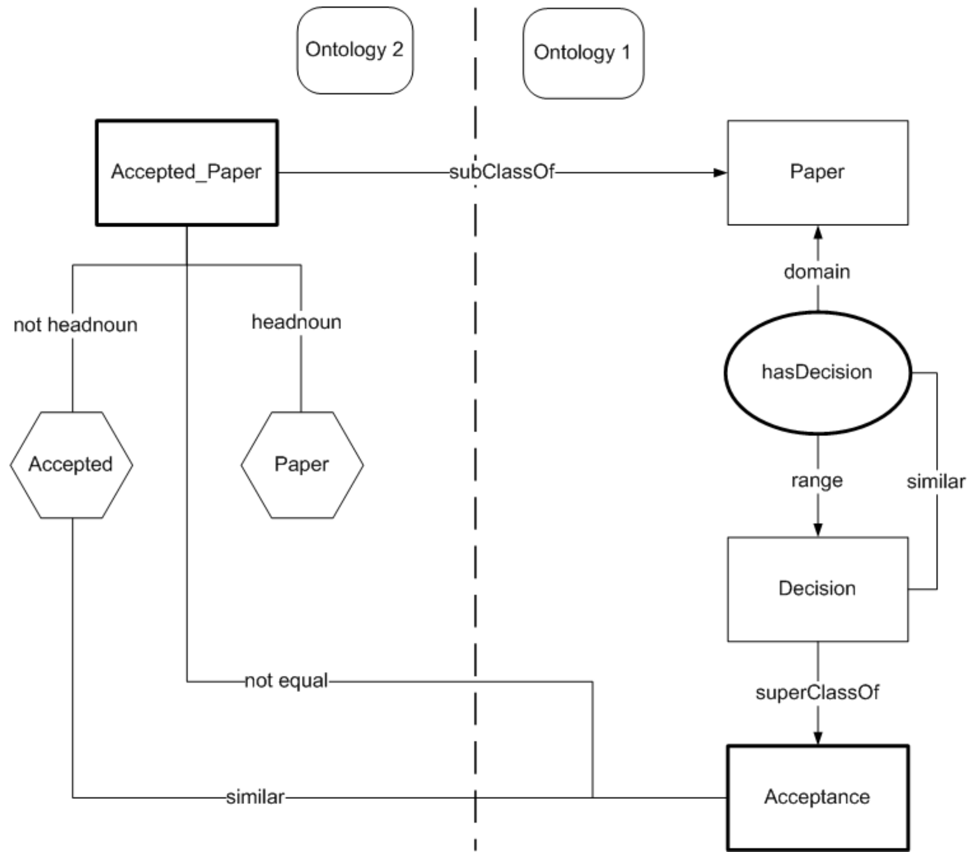

Figure 3.3: Correspondence of type Qualified Restriction between *Accepted_Paper* and *hasDecision* with range *Acceptance*. *Accepted_Paper* can be found in OAEI conference ekaw, *hasDecision* and *Acceptance* in OAEI conference cmt.

rithm 3. First the algorithm iterates through all datatype properties of $\mathcal{O}_1$. Further all classes of $\mathcal{O}_2$ are examined. Given $1\#p$ its domain $1\#d$ and range $1\#r$ is determined. Afterwards it is checked whether $1\#p$ and the reduced name of $2\#c$ are similar to each other. Continuing, the superclasses of $2\#c$ are assigned, especially the superclasses of $\mathcal{O}_1$ which have been deduced from the alignment. Moreover $1\#s$ has to be a subclass of the domain $1\#d$ and also the range $1\#r$ should be boolean. In general every finite range is imaginable but at the moment only the boolean range is implemented because it is the only one used within the example ontologies. If a non-boolean range is available the algorithm has to be expanded to be able to decide which value is adequate for which class. The last condition checks the similarity of $1\#p$ and the first part of the reduced name of $2\#c$. Having $2\#c$ where the first part of the name is similar to $1\#p$, a correspondence is found with the value true, else with value false. Hence the similarity of $2\#c$ and $1\#p$ just indicate which value is determined to be part of the correspondence. The found correspondence is: $\forall$x ($2\#c$(x) $\leftrightarrow$ $\exists$y ($1\#p$(x,y) $\wedge$ y='true')) or $\forall$x ($2\#c$(x) $\leftrightarrow$ $\exists$y ($1\#p$(x,y).

For getting a better understanding in Figure 3.4 an example is illustrated. On the left side the class *Early-Registered_Participant* out of $\mathcal{O}_1$ can be split up into "Early-Registered" and "Participant". For "Participant" a class in $\mathcal{O}_2$ exists particular *Participant* which is similar to it. Early-Registered is therefore the reduced class name because it is not the part having the most similar class in $\mathcal{O}_2$. Additionally *Early-Registered* is similar to the property *earlyRegistration*. A superclass of *Early-Registered_Participant* is *Participant* in $\mathcal{O}_2$ which is also the domain of the property as it is required. Furthermore the range of the property is boolean and the first part of the class, here Early, is similar to the property name. Now a complex correspondence of type *Value Restriction* is found: $\forall$x (*Early-Registered_Participant(x)* $\leftrightarrow$ $\exists$y *earlyRegistration(x,y)* $\wedge$ y='true')).

3.4 Property chain

Two properties which can be applied successively might express the same as one property. Algorithm 4 describes the way of finding such correspondences. It is restricted to the datatype property for the isolated property because no other examples are contained in the ontologies. At the beginning a datatype property $1\#p$ out of $\mathcal{O}_1$ and an object property $2\#p$ of $\mathcal{O}_2$ are given by iterating through all corresponding properties. Further the domains and ranges are computed, $1\#r$ range and $1\#d$ domain of $1\#p$, $2\#r$ range and $2\#d$ domain of $2\#p$. Further the names of $1\#p$ and $2\#p$ should be similar and also $2\#d$ a subclass of $1\#d$. The last con-

Algorithm 3 Find Value Restriction correspondences

VALUE RESTRICTION($\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$)

1: **for** $1\#p$ in DATATYPEPROPERTIES($\mathcal{O}_1$) **do**
2:     **for** $2\#c$ in CLASSES($\mathcal{O}_2$) **do**
3:         $1\#r \leftarrow$ RANGE($1\#p$, $\mathcal{O}_1$)
4:         $1\#d \leftarrow$ DOMAIN($1\#p$, $\mathcal{O}_1$)
5:         **if** ISSIMILAR($1\#p$, REDUCEDNAME($2\#c$, $\mathcal{O}_2$)) **then**
6:             **for** $1\#s$ in ALIGNEDSUPERCLASSES($2\#c$, $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$) **do**
7:                 **if** $1\#s \sqsubseteq 1\#d$ **and** $1\#r =$ 'boolean' **then**
8:                     **if** ISSIMILAR(FIRSTPARTOFNAME(REDUCEDNAME($2\#c$)), $1\#p$) **then**
9:                         ▷ *create* $\forall x\ (2\#c(x) \leftrightarrow \exists y\ (1\#p(x,y) \wedge y=\text{'true'}))$
10:                     **else**
11:                         ▷ *create* $\forall x\ (2\#c(x) \leftrightarrow \exists y\ (1\#p(x,y) \wedge y=\text{'false'}))$
12:                     **end if**
13:                 **end if**
14:             **end for**
15:         **end if**
16:     **end for**
17: **end for**

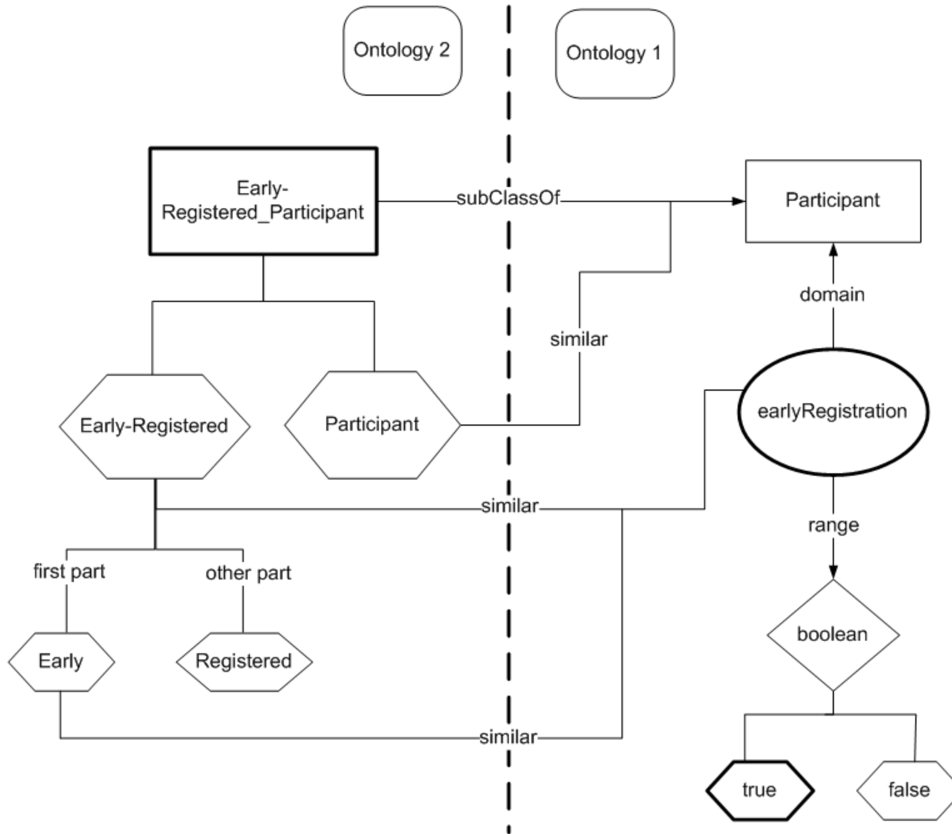

Figure 3.4: Correspondence of type Value Restriction between *Early-Registered_Participant* and *earlyRegistration* with value true. *Early-Registered_Participant* can be found in OAEI conference ekaw, *earlyRegistration* in OAEI conference confOf.

dition can be checked by inspecting the alignment. Now the datatype properties of $\mathcal{O}_2$ are examined. The range $2\#r$ of the $2\#p$ has to be a superclass of the domain $2\#d'$ of the $2\#p'$ otherwise the properties cannot be successively applied. Moreover the ranges $1\#r$ and $2\#r'$ have to be compatible, this means one datatype can be expressed in the other datatype. For example string is compatible to every other datatype because everything can be expressed as a string. On the other hand boolean and date are not compatible because a boolean can never be transformed into a date. If two datatype properties $1\#p$ and $2\#p'$ are given which satisfies the requirements above, it has to be decided whether the two properties executed one after another really express the same as the other property. This can be accomplished by inspecting the name of $2\#p'$ contained in the chain. If the name is "name" or contains the name of $1\#p$, a correspondence is found. Taking all datatype properties called "name" may cause some wrong results, but often really the name of an object is meant, e.g. a datatype property called author normally describes the name of the author although the word "name" is not contained in the name of the property.

Illustrating the algorithm, Figure 3.5 shows an example found in the ontologies examined. On the left side the separated property can be seen, on the right the property chain. The properties *hasJournal*, datatype property, and *journal*, object property, are namely very similar. Also the domain of *hasJournal* which is *Entry* is a superclass of the domain *Article* from *journal*. Further the datatype property *name* in $\mathcal{O}_2$ can be found and its domain *Journal* is simultaneously the range of *journal*. Both datatype properties *hasJournal* and *name* has the range String and of course String is compatible to String. Having the property *name* also automatically fulfills the last requirement. In this case it is meaningful because *hasJournal* indicated the name of the journal in which a certain entry is published. Applying the property chain of *journal* and *name* also relates an entry, here especially article, to the name of the journal in which it is published. Formally the correspondence can be expressed as following: $\forall$x,y (*hasJournal(x,y)* $\leftrightarrow$ $\exists$z (*journal(x,z)* $\wedge$ *name(z,y)*)).

Algorithm 4 Find Property Chain correspondences

PROPERTY CHAIN($\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{A}$)

1: **for** $1\#p$ in DATATYPEPROPERTIES($\mathcal{O}_1$) **do**
2: **for** $2\#p$ in OBJECTPROPERTIES($\mathcal{O}_2$) **do**
3: $1\#r \leftarrow$ RANGE($1\#p$, $\mathcal{O}_1$)
4: $1\#d \leftarrow$ DOMAIN($1\#p$, $\mathcal{O}_1$)
5: $2\#d \leftarrow$ DOMAIN($2\#p$, $\mathcal{O}_2$)
6: $2\#r \leftarrow$ RANGE($2\#p$, $\mathcal{O}_2$)
7: **if** ISSIMILAR($1\#p$, $2\#p$) AND $2\#d \sqsubseteq$1#d **then**
8: **for** $2\#p'$ in DATATYPEPROPERTIES($\mathcal{O}_2$) **do**
9: $2\#d' \leftarrow$ DOMAIN($2\#p'$, $\mathcal{O}_2$)
10: $2\#r' \leftarrow$ DOMAIN($2\#p'$, $\mathcal{O}_2$)
11: **if** $2\#d' \sqsubseteq 2\#r$ AND ISCOMPATIBLE($1\#r$, $2\#r'$) AND (CONTAINS($2\#r'$, 'name') OR CONTAINS($2\#r'$, $1\#r$)) **then**
12: ▷ *create* $\forall x,y\ (1\#p(x) \leftrightarrow \exists z\ (2\#p(x,z) \wedge 2\#p'(z,y)))$
13: **end if**
14: **end for**
15: **end if**
16: **end for**
17: **end for**

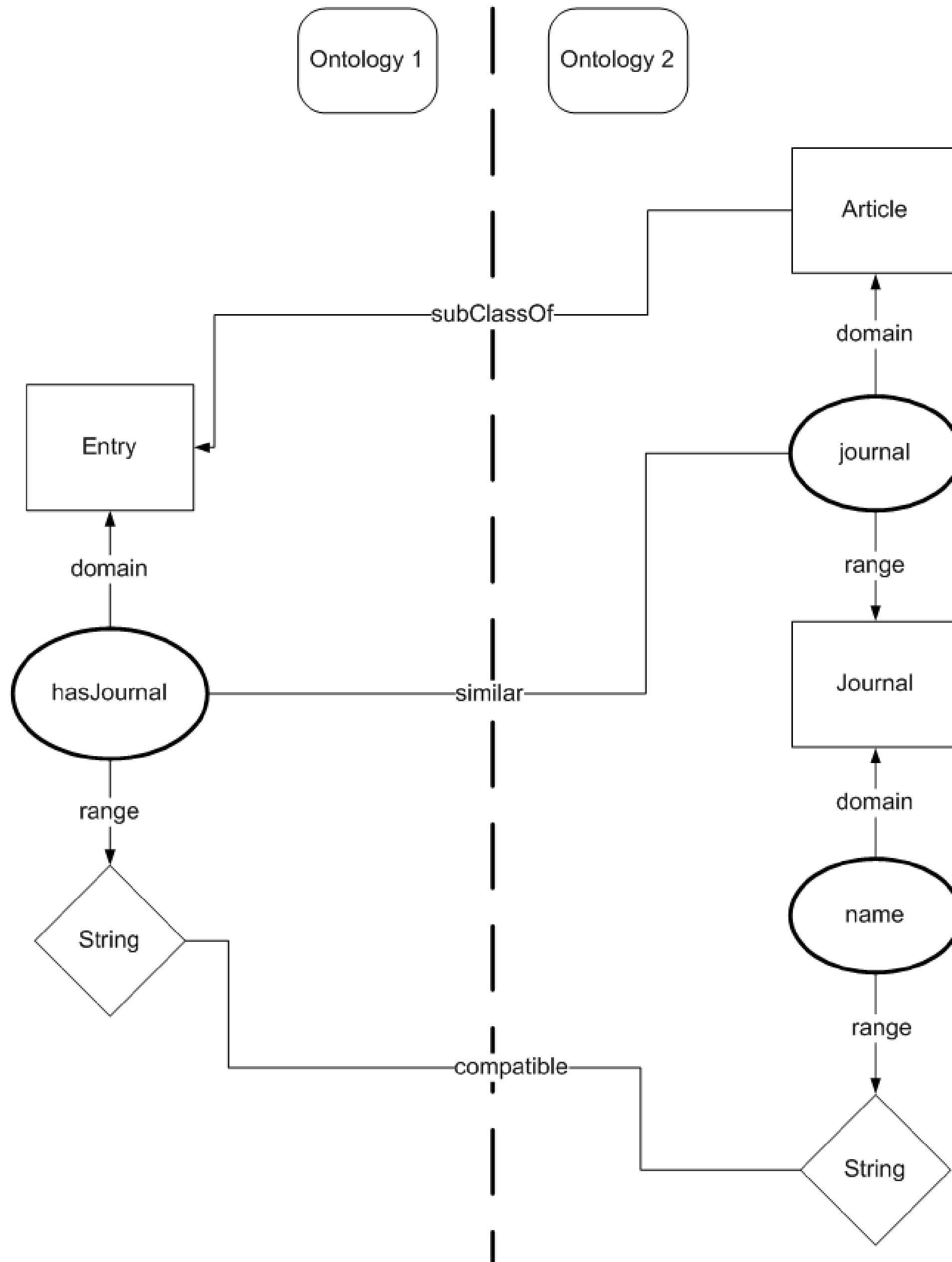

Figure 3.5: Correspondence of type property chain between *hasJournal* and the chain of *journal* and *name*. Having a property called name satisfies the requirement given in the algorithm for the second datatype property which is not particular shown in the picture. The property *hasJournal* is contained in OAEI benchmark 301, *journal* and *name* in OAEI benchmark 101.

Chapter 4

Experiments

After giving an introduction into ontology matching and the required background knowledge, different types of complex correspondences have been described as well as the way of finding them. Also the need of such correspondences has been discussed and demonstrated. For implementing some types of correspondences a decision had to be taken which types should be selected based on different criteria, e.g. the number of appearances in the examined ontologies. All created algorithms out of the ontologies have been illustrated and explained with an example. Now the results of the algorithms applied on some ontologies are pointed out and evaluated. Additionally a short outline of the implementation and the external tools used is given.

4.1 Implementation

In this chapter the implementation of the algorithms is described. The intention is to give a short overview but not a complete analysis with the functionality and the structure. Further the external tools which have been integrated into the system are presented to get an impression of their function. Everything is implemented with the Java programming language, release 1.6. Figure 4.1 shows the packages, classes with attributes, methods and relations between them in a very simplified way. Rounded rectangles stand for packages, all other rectangles, which are further divided, for classes and ellipses for external tools. Several classes are contained in a package because they share or compute the same things. Every class is partitioned into three parts: name, attributes and methods. This separation is similar to the well-known UML notation, but not as exactly and detailed as UML is. Name is just the name of the class, attributes are characteristics of the class and mostly build the content, methods are usually used to compute the content. Additionally

lines with filled arrows illustrate a relation between classes or packages by means of a class/package uses methods from another class in the package or instantiate a class. On the other hand lines with a non filled arrow define an inheritance class hierarchy where every subclass inherits all methods and attributes from its superclass.
First the used external tools OWL API [1], Pellet [22], Matcher and parts of Alcomo[1] for alignments are explained. OWL API is a java implementation for the Web Ontology Language OWL. With the OWL API, release 2.2.0, it is possible to access the content of an ontology for further computations. Every package makes use of this tool because of accessing parts of the ontology such as classes or properties. Beside OWL, API Pellet, version 2.00 rc4, an OWL reasoner is integrated to gather additional information. For example if a property has a domain $\mathcal{D}$ and $\mathcal{D}$ has the subclass $\mathcal{C}$, OWL API only indicates $\mathcal{D}$ as domain and not $\mathcal{C}$ although logically $\mathcal{C}$ is of course part of the domain. Pellet can find such auxiliary information based on the class hierarchy. Having all this information is necessary to draw further conclusions when finding correspondences. Further every algorithm requires values for the similarity of names. To compute these similarities a matcher with linguistic approaches based on a similarity measure called Levenshtein [15] is utilized. In general all algorithms are not restricted to a certain matcher, only the values are needed without exactly knowing how they are calculated. It is only important that the matcher uses a linguistic approach. For reading in an alignment in XML or text format some implementations from the project Alcomo have been used.
Now the packages and classes are explained and their relations among each other. In the upper left corner the package complexMapping with its classes ComplexMappingException and ComplexMapping can be found. It defines the interface to the user of the program and every input/output is dealt with here. ComplexMappingException presents the exception which is thrown if an error occurs. Every other exception is intercepted so classes only have to integrate this exception. Additional information about the error are also outputted to know more precisely why this error occurs. The constants for identifying are used to identify the given error, with a constructor a new instance of a ComplexMappingException can be created and toString only defines the output. The main class is called ComplexMapping and contains the main method. If someone uses the program, the input is committed to the main method where the data is saved and all further computations are started with the help of the constructors and the methods createAlignment and writeAlignment as well. Getting a complex alignment requires two ontologies, a reference alignment and, if desired, a path to a file where the alignment should be

[1]The modules used in [17] have been provided.

written to. No matter which variant is demanded, every correspondence is shown as output on the console. Further the package correspondence, which in general handles the complex correspondences, contains the classes Correspondence and CorrespondenceWriter. The most important class here is Correspondence which includes the content of a found correspondence, e.g. the included classes and properties. Of course some other packages/classes implement this class because of creating correspondences in the algorithms or for the output in the class ComplexMapping. The methods toString and toShorterString represent the output of an individual correspondence where toShorterString does not hold the whole URI but a shorter version for better readability. Writing a whole alignment is realized by the class CorrespondenceWriter which writes a file based on its correspondences. Next, the package called utility, in the upper right corner, includes the classes Ontology and OntologyAlignment to save all data about an ontology or an alignment. Having the Ontology class makes it easier to access all data of an ontology like classes or properties without yet again inspecting the ontology. Methods of this class are a constructor, buildOntology which is called by the constructor to create and save all data and finally getters to customize the information for other classes. OntologyAlignment saves the file path of the alignment and guarantees access to it. Many packages make use of these classes to get all needed information for further computations. Additionally the package reasoner in the bottom right corner implements the reasoning and everything which is needed for this like reading in the alignment. Reasoner is the superclass of ReasonerOneOntology and ReasonerTwoOntologies and contains all attributes and methods both subclasses share to avoid code replication. On one hand ReasonerOneOntology can just perform reasoning on one ontology and does not need another ontology or an alignment in this case. On the other hand ReasonerTwoOntologies requires two ontologies and an alignment, which is read in by the Alcomo component, for accessing both ontologies. If the request is only limited on one ontology, it would be meaningless to deliver two ontologies and an alignment. Both classes have a method called getTypeClasses where one can decide which classes should be returned, subclasses, superclasses, ancestor and descendant are available. This method internally call getClasses in its superclass. Having all packages described above, a foundation for implementing the algorithms is now available. Finally the package correspondenceComputation contains a class for every algorithm and some auxiliary classes to simplify the computations. Every class computing a certain type of complex correspondence contains the computation method and later also the computed correspondences. ComputingUtility includes methods, like getDomain/getRange or similarity value of two names, used by several computation classes. The last class in this package is called ComputationSetting and contains variables which can be modified. For example the decision whether two classes in a certain algorithm are

similar is defined here.

Furthermore some information about the resulting alignment is important to understand which correspondences it contains and why. First the order of the ontologies is not distinctive for the complex alignment. Every computation algorithm to find correspondences is executed in both directions, once with the given order of the ontologies and later vice versa. Having an order independent alignment seems to be a good way because of getting symmetric result which is often useful. Additionally not every found correspondence is available in the alignment. Correspondences are only added if the entity on the left side, the side where only one entity can be found, is not already in the reference alignment. As an exception correspondences between object and datatype properties are not considered further because otherwise correspondences of type property chain are not found because of comparing the different property types in the alignments. Although some correspondence might be correct and they just describe an entity in an additional way, they are deleted because for the same entity more than one description is not necessary in this context. If two or more correspondences of the same type are found and one correspondence can be considered as the most general one, all other correspondences are not adopted into the alignment. For example two correspondences of type *Unqualified Restriction*: $\forall$x (*Reviewed_Paper(x)* $\leftrightarrow$ $\exists$y (*reviewOfPaper(y,x)* $\wedge$ *Paper(x)*)) and $\forall$x (*Reviewed_Paper(x)* $\leftrightarrow$ $\exists$y (*reviewOfPaper(y,x)* $\wedge$ *Submitted_Paper(x)*)) where *Reviewed_Paper* is a class in $\mathcal{O}_1$, all other entities out of $\mathcal{O}_2$. Moreover *Paper* is a superclass of *Submitted_Paper*. In the alignment only $\forall$x (*Reviewed_Paper(x)* $\leftrightarrow$ $\exists$y (*reviewOfPaper(y,x)* $\wedge$ *Paper(x)*)) can be found because of being the more general one. For every entity only one description in terms of a correspondence should be found, neither another complex correspondence nor an existing plain equivalence one in the alignment. If several complex correspondences of the same type are available and none of them is more general than other ones, the similarity values calculated while computing the correspondences are inspected. If the sum of the values from one correspondence is greater than the other ones, this correspondence is chosen and all other correspondences of that kind are deleted. A difference between the sums of similarity values is not always found. In this case all possible correspondences are in the alignment. Also different types of complex correspondences with the same entity on the left side are contained in the complex alignment later. Altogether the intension is a 1:1 mapping but it is not always possible because of not having further criteria to decide which is the right one. Hence an alignment can contain several correspondences with the same left side.

All correspondences in the alignment are also in first-order-logic like every correspondence before. This is not a standard like XML and is not yet supported by any

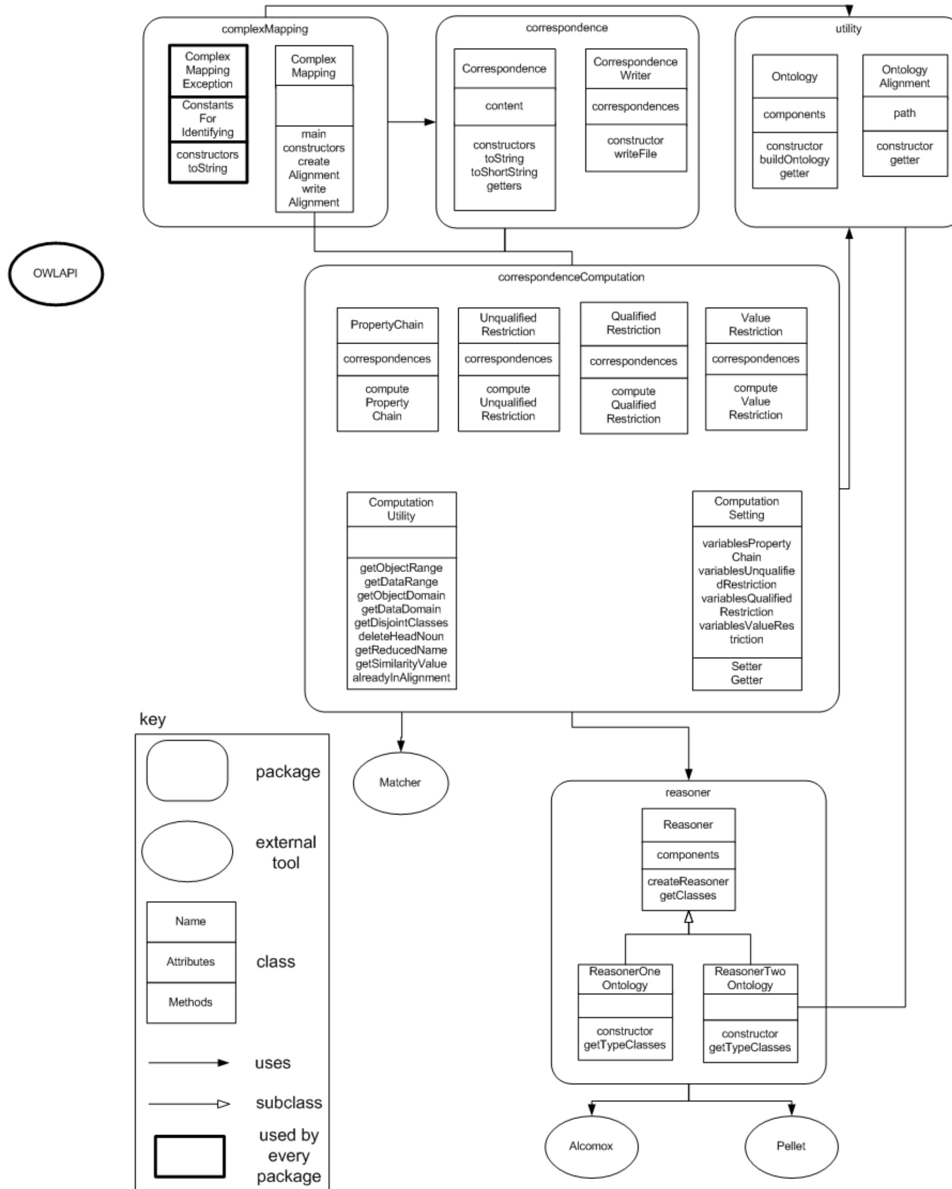

Figure 4.1: Implementation overview with packages, classes and external tools

	cmt	confOf	ekaw	iasted	sigkdd	sofsem	crs	openconf	pcs
cmt	-	(1)	(1)	(1)	(1)	(2)	(2)	(2)	(2)
confOf	-	-	(1)	(1)	(1)	-	(2)	(2)	(2)
ekaw	-	-	-	(1)	(1)	-	-	(2)	(2)
iasted	-	-	-	-	(1)	-	-	-	-
sigkdd	-	-	-	-	-	-	-	-	-
sofsem	-	(2)	(2)	-	(2)	-	(2)	(2)	(2)
crs	-	-	(2)	-	(2)	-	-	(2)	(2)
openconf	-	-	-	-	(2)	-	-	-	(2)
pcs	-	-	-	-	(2)	-	-	-	-

Table 4.1: Considered conference ontologies for testing

tool which deals with reading in alignments. But it would not be too difficult to transform the output into an standardized format.

4.2 Settings

After completing the implementation it is possible to test the application. In this section the settings for all executed tests are specified to understand the results given in the next section. All tests have been performed on the benchmark, conference and on a second conference set. The first two have been examined while looking for examples of complex correspondences. Having a set ontologies which has not been inspected while looking for examples indicates whether the correspondences are general enough or not. If the algorithms are able to find the correspondences in the second conference set, it shows that they are not restricted to specific patterns which were only available in the other datasets. Although both conference sets, first one the OAEI alignment 2008 set and second one the alignment AAAI 2007 set, describe the same issue and share some ontologies, they have been developed independently. This means not every ontology has been created by just one person, instead the ontologies have been created by different persons without agreement.

Below all pairs of ontologies are listed to understand the results following in the next section, Section 4.3. Not every possible pair of ontologies has been taken into account because a reference alignment does not exist for every pair but is required by the algorithms. In Table 4.1 all confer

In the benchmark set only the following pairs have been regarded: 101-301, 101-302, 101-303, 101-304. In general, considering more ontologies was not pos-

sible because either no reference alignment has been available or the ontologies have not been expressive enough, e.g. they contain too little entities to be suitable for computing complex correspondences.

As mentioned in Section 4.1, all thresholds for similarity are implemented as variables which can be modified. In the beginning all variables had to be set to fixed values to start testing. In general the thresholds should not be too high to avoid algorithms which only find the previously located examples and neither incorrect nor other right ones. To show how the results vary, three different sets of thresholds have been tested. First relatively low thresholds, second mid thresholds and third high thresholds. Thereby it becomes obvious how the choice of the thresholds affect the results. In the beginning every threshold has been set up to the same value. The three types *Value Restriction*, *Unqualified Restriction* and *Property Chain* provided acceptable results in contrast to *Qualified Restriction* where the results were not so good. Far more than three out of four correspondences were wrong whereby the guide line was to have at least one quarter correct ones. This relatively high error rate for the *Qualified Restriction* is explainable because the two variables which can be set in the corresponding algorithm are not very restricting. One variable adjusts the threshold which determines if a similarity between the head noun of a class name in $\mathcal{O}_1$ to a complete class name in $\mathcal{O}_2$ is available and the other one decides if an object property is similar to its range. Beside the fact that classes are the main entities, every class is usually similar to some classes in the other ontology, especially if similarity is achieved by having a low similarity value. Also the constraint object property and its range has to be similar is within easy reach because many properties are named like their range because they map an entity onto a specific range. For example a property is called *name* because it relates an entity to its name. Different from *Qualified Restriction* all other types involve stronger constraints or operate more on properties, of which there are usually not so many contained in an ontology, but less on classes. Finding a correspondence of type *Value Restriction* is more difficult because a datatype property with a finite range, here especially a boolean range, has to be contained in one ontology. After determining the problem with *Qualified Restriction*, all variables can be modified such that the values are possibly low but not more than three out of four correspondences of every type are wrong. Of course these settings are fitted and maybe other ontologies need other values which is not completely excludable. Having the set of low thresholds, the other two threshold sets have to be defined. For better comparison every variable of one type has the same threshold like in the set with the low threshold and also the gap between the thresholds of *Value Restriction*, *Unqualified Restriction*, *Property Chain* and *Qualified Restriction* is maintained. To see how the results change but not having to high thresholds, the mid thresholds have been increased by 0.1 in comparison to the low thresholds. Finally the high thresholds

have be determined which should be high enough to recognize differences.

- Low thresholds: 0.6 for Qualified Restriction, 0.5 all others
- Mid thresholds: 0.7 for Qualified Restriction, 0.6 all others
- High thresholds: 0.9 for Qualified Restriction, 0.8 all others

Summarizing, all tests are based on the ontologies listed above and have been executed with three sets of different thresholds: low, mid and high.

4.3 Results

After explaining the functionality of the algorithms and how they have been implemented, the settings have been defined which determine a testing environment where tests can be started. The results of the executed tests are given in this section to get an impression how useful and successful the ideas are de facto. Below, three tables are shown with the results of the test runs. Table 4.2 illustrates the findings with low thresholds, Table 4.3 with mid thresholds and Table 4.4 with high thresholds. UQR is a abbreviation for *Unqualified Restriction*, QR for *Qualified Restriction*, VR for *Value Restriction* and PC for *Property Chain*. Every row shows the number of found correspondences of a certain set of ontologies. Conference 1 are the Conference ontologies which have been examined while trying to find examples and Conference 2 the second Conference ontologies which have not been inspected before. Also the sum of correspondences per type can be found in a row. The last row shows for every type the number of correspondences found in all three ontology sets. All found correspondences are further divided into right and wrong correspondences which correspond to the two columns Right and Wrong. Moreover another column called Size Input Alignment declares how many correspondences have been found in the given reference alignments. Thereby only correspondences between classes have been counted to have a common foundation, because the Benchmark alignments sometimes contain correspondences between properties different from the Conference alignments. For every set of ontologies only one number is available and is composed of the number of correspondences in every particular input alignment. Specifying the size of the reference alignments show the amount of complex correspondences compared with the plain equivalence correspondences in the alignments.

Looking at Table 4.2 most conspicuous might be the number of right *Property Chain* and of wrong Q*ualified Restriction* correspondences. In Section 4.2 the reason for having a high error rate while searching *Qualified Restriction* correspondences has been already discussed. In the benchmark ontologies only *Property*

	Right					Wrong					Size Input Alignment
Ontologies	UQR	QR	VR	PC	$\sum$	UQR	QR	VR	PC	$\sum$	
Conference 1	1	2	2	0	5	3	10	0	5	18	102
Conference 2	1	2	0	0	3	1	4	0	11	16	173
Benchmark	0	0	0	17	17	0	0	0	8	8	94
$\sum$	**2**	**4**	**2**	**17**	**25**	**4**	**14**	**0**	**24**	**42**	**369**

Table 4.2: Results with the low thresholds.

Chain occurs and no other types because because they are different from the Conference ontologies. Instead correct *Property Chain* correspondences can only be found in these ontologies. Although the Conference 2 ontologies have not been inspected before, complex correspondences of type *Qualified* and *Unqualified Restriction* are found. Only correspondences of type *Value Restriction* are not found, different from Conference 1, but this type is special and if no datatype property with a boolean range is available in the ontologies, such a correspondence cannot be found. Therefore in Conference 2 not so many wrong *Qualified* and *Unqualified Restriction* correspondences are declared.

	Right					Wrong					Size Input Alignment
Ontologies	UQR	QR	VR	PC	$\sum$	UQR	QR	VR	PC	$\sum$	
Conference 1	1	2	2	0	5	2	3	0	2	7	102
Conference 2	1	2	0	0	3	1	2	0	5	8	173
Benchmark	0	0	0	17	17	0	0	0	6	6	94
$\sum$	**2**	**4**	**2**	**17**	**25**	**3**	**5**	**0**	**13**	**21**	**369**

Table 4.3: Results with the mid thresholds.

Based on the results of the test with low thresholds comparisons with the mid thresholds in Table 4.3 can be made. All correct complex correspondences are also found with the increased thresholds. Only the number of wrong correspondences has changed, especially the number of *Qualified Restriction* and *Property Chain* has been reduced a lot. These thresholds seem to be very good but they are not as general as the low thresholds and maybe other ontologies contain correct correspondences which are not found with the mid thresholds different from the low thresholds. Increasing the thresholds lead in this case to a improvement of the results hence a increasing has been applied to see if the results are getting better again.

	Right					Wrong					Size Input Alignment
Ontologies	UQR	QR	VR	PC	$\sum$	UQR	QR	VR	PC	$\sum$	
Conference 1	1	0	0	0	1	1	2	0	0	3	102
Conference 2	0	0	0	0	0	1	0	0	1	2	173
Benchmark	0	0	0	17	17	0	0	0	4	4	94
$\sum$	**1**	**0**	**0**	**17**	**18**	**2**	**2**	**0**	**5**	**9**	**369**

Table 4.4: Results with the high thresholds.

In Table 4.4 the results of the test with the highest threshold are shown. Beside the number of wrong, now also the number of correct correspondences has been reduced. Not as maybe imagined only the number of incorrect correspondences lowered and increasing the thresholds does not improve the result. Choosing very high thresholds therefore is not the solution to get all correct correspondences and also reduce the error rate.

Altogether the three different tests exemplify the effect on the results if the thresholds are changed. Setting the thresholds to a high level does not increase the number of correct correspondences and simultaneously reduce the number of incorrect ones. With low thresholds the generality and the chance to find not yet known examples raises but on the other hand also the error rate raises. A balanced solution seems to be best except of the generality and the chance to find new examples. Further every threshold can be modified separated without taking the other thresholds of the same or different type into account. Thereby it might be possible to get better results but with exactly fitted thresholds the chance to find new correspondences decreases a lot. That is why in all tests above only one value has been determined for all thresholds corresponding to the same type of complex correspondence.

The number of occurrences of these types in the ontologies should not be criticized based on these results because the correspondences may help to get answers to several questions.Without knowing about these correspondences it would not be possible. For example someone wants to know which papers have been accepted on a certain conference. Ontology $\mathcal{O}_1$ implements the class *Accepted_Paper* and ontology $\mathcal{O}_2$ the object property *hasDecision* with range *Decision* and *Acceptance* is a subclass of *Decision*. This example was already mentioned while explaining the algorithms in Chapter 3. If the information about the papers are saved in these both ontologies whereby not every paper is in both ontologies. Without the com-

plex correspondence of type *Qualified Restriction* between *Accepted_Paper* and *hasDecision Acceptance* it is not possible to really get all accepted papers.

To sum up the results, although if not a huge amount of complex correspondences has been found in the ontologies, the fact that in an unregarded set of ontologies, Conference 2, these correspondences have also been found shows the boundlessness of the types on special ontologies. Additionally detecting complex correspondences makes it possible to gather new information from ontologies which have not been available before.

Chapter 5

Summary

5.1 Conclusion

Avoiding heterogeneity of ontologies is not indeed possible as soon as different persons create them. Imagine two humans both independently building an ontology describing a company. Naturally the ontologies differ in some facets and are not completely the same. While comparing the ontologies one probably determines that not every entity correspond to exactly one entity of the other ontology. Therefore it might be necessary to compare one entity with a complex term composed of several entities. Nowadays detecting complex correspondences with state of the art matching techniques is not possible in a way as it is necessary for ontology matching. Only a few matching systems take complex correspondences, but mostly just non-equivalence correspondences, into account although the need is obvious. To get an impression, several types of complex correspondences have been described. They have been detected in state of the art test sets of the OAEI. It turned out that four patterns occurred relatively frequent, therefore algorithms have been developed to automatically find correspondences of these types. Additionally examples have been shown to get a better comprehension of the patterns and their algorithms. More precise *Unqualified Restriction*, *Qualified Restriction*, *Value Restriction* and *Property Chain* are the types which have been particularly examined. These four seemed to be most promising and not restricted to some ontologies rather they describe different forms of modeling an issue.

The algorithms should be as simple as possible and just a few requirements have to be accomplished to find a correspondence of a certain type. Due to the implementation it is feasible to find complex correspondences automatically. Moreover the implemented algorithms have been tested to see if they work and how many complex correspondences can be found. Not only the ontologies inspected

while finding examples have been tested, also some unknown ones. Analyzing the results, it became obvious that every example manually found has been also detected while testing the implemented algorithms with the known ontology sets. Additionally some further examples have been found which were not predictable. Also complex correspondences in the unknown ontologies have been found which shows the generality of the types being not restricted to particular ontologies. With the help of complex correspondences it is possible to get answers to queries which could not be found before. Imagine data distributed to ontologies which are only related to each other through complex correspondences. Without knowing the equivalences it is not be possible to get all requested data with only one query. Instead a query for every ontology fitted to its structure and entities would be necessary. Actually this is not very meaningful and convenient for a user especially for an inexperienced one.

All together ignoring complex types reduces the information contained in an alignment and therefore also information about overlapping concepts in the ontologies. In an extreme case the alignment might be empty even if the ontologies describe the same domain of interest. Due to all reasons mentioned in this thesis the need for complex correspondences and their further development becomes clearly apparent.

5.2 Future Work

There are several open problems for future work. The first problem concerns the implemented types and their implementation. At present as a strictly 1:1 mapping is not available, it may happen that an entity of one complex correspondence also occurs in another correspondence. If two correspondences share an entity the similarity values the algorithm computed are checked and if the values from one correspondence are higher, this correspondence is chosen and can be found in the alignment later. In some cases the values are exactly the same and no decision can be taken. Considering techniques to come to a decision or user interaction is necessary to get a 1:1 mapping.

Furthermore the alignment of complex correspondences is not in a standardized form like XML or text format. Conventional alignment readers are not able to read in the alignment and even if the alignment would be in a suitable format it may cause some problems because not just two entities may be a part of the correspondence. Additionally the algorithms could be checked to refine or generalize them because not every algorithm might be as general as possible and changing

some statements or adding new ones could enhance the results. Especially the type *Value Restriction* takes only boolean range into account so far whereas every finite datatype range is imaginable.
Apart from the implementation, more ontologies with an existing alignment are needed to get more testing results. If more data is available it would be possible to adapt the algorithms and to get an expressive estimation for every implemented type. Another task for future would be the implementation of all types which have been found. In general finding new types of complex correspondences and implementing them is required if complex correspondences should be taken into account. The set of presented types is only an extract of all imaginable types of correspondences. There is more awareness needed concerning the complexity of correspondences in ontology matching to achieve the presented ideas. As long as no one is conscious of complex types, new ones will not be found and existing matching systems will not be able to compute correspondences of that kind.

Bibliography

[1] Sean Bechhofer an Phillip Lord and Raphael Volz. Cooking the semantic web with the owl api. 2nd International Semantic Web Conference, ISWC, 2003.

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284 (5):34–43, 2001.

[3] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language(xml) 1.0, 1998.

[4] Caterina Caracciolo, Jérôme Euzenat, Laura Hollink, Ryutaro Ichise, Antoine Isaac, Véronique Malaisé, Christian Meilicke, Juan Pane, Pavel Shvaiko, Heiner Stuckenschmidt, Ondřej Šváb Zamazal, and Vojtěch Svátek. Results of the ontology alignment evaluation initiative 2008. In *Ontology Matching 2008 proceedings*, pages 73–119, 2008.

[5] Jérôme David. Aroma results for oaei 2008. In *Ontology Matching 2008 proceedings*, pages 128–131, 2008.

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

[7] Christiane Fellbaum. Wordnet: An electronic lexical database, 1998.

[8] J. Gracia and E. Mena. Ontology matching with cider: Evaluation report for the oaei. In *Ontology Matching 2008 proceedings*, 2008.

[9] Fayçal Hamdi, Haïfa Zargayouna, Brigitte Safar, and Chantal Reynaud. Taxomap in the oaei 2008 alignment contest. In *Ontology Matching 2008 proceedings*, pages 206–213, 2008.

[10] Wei Hu, Yuanyuan Zhao, Dan Li, Gong Cheng, Honghan Wu, and Yuzhong Qu. Falcon-ao: Results for oaei 2007. In *Ontology Matching 2007 proceedings*, pages 170–178, 2007.

[11] Yves R. Jean-Mary and Mansur R. Kabuka. Asmov: results for oaei 2008. In *Ontology Matching 2008 proceedings*, pages 132–139, 2008.

[12] Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf): Concepts and abstract syntax, 2004.

[13] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. The protégé owl plugin: An open development environment for semantic web applications. ISWC, 2004.

[14] Patrick Lambrix, He Tan, and Qiang Liu. Sambo and sambodtf results for the ontology alignment evaluation initiative 2008. In *Ontology Matching 2008 proceedings*, pages 190–198, 2008.

[15] Vladimir I. Levenshtein. Binary codes capable of correcting deletions and insertions and reversals. *Doklady Akademii Nauk SSSR*, pages 845–848, 1965. In Russian. English Translation in Soviet Physics Doklady, 10(8) p. 707710, 1966.

[16] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview, 2004.

[17] Christian Meilicke and Heiner Stuckenschmidt. Applying logical constraints to ontology matching. In *Proceedings of the 30th German Conference on Artificial Intelligence(KI-07)*, pages 99–113. Springer, 2007.

[18] Han Qin, Dejing Dou, and Paea LePendu. Discovering executable semantic mappings between ontologies. *In On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, pages 832–849, 2007.

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

[20] M. Sabou, M. d'Aquin, and E. Motta. Exploring the semanticweb as background knowledge for ontology matching. *Journal on Data Semantics*, XI, 2008.

[21] Marta Sabou and Jorge Gracia. Spider: bringing non-equivalence mappings to oaei. In *Ontology Matching 2008 proceedings*, pages 199–205, 2008.

[22] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner.

[23] Heiner Stuckenschmidt, Livia Predoiu, and Christian Meilicke. Learning complex ontology alignments a challenge for ilp research. In *Proceedings of the 18th International Conference on Inductive Logic Programming*, 2008.

[24] Ondřej Šváb, Vojtěch Svátek, Petr Berka, Dušan Rak, and Petr Tomášek. Ontofarm: Towards an experimental collection of parallel ontologies. In *Poster Proceedings of the International Semantic Web Conference*, 2005.

Appendix A

Program Code / Resources

Beside the thesis, also a CD is attached, containing the program code and executable files to compute complex mappings. Following folders are on the CD: *Code*, *Javadoc*, *Ontologies*, *Complex_Mapping_CD*, *Complex_Mapping_Windows*, *Complex_Mapping_Linux*. The folder *Code* just contains the java code and can be imported for example into the IDE Eclipse. In *Javadoc* the corresponding javadoc can be found. Every ontology and alignment used during the testing process is contained in *Ontologies* as well as their complex mappings. In *Complex_Mapping_CD*, *Complex_Mapping_Windows* and *Complex_Mapping_Linux* the executable files are located. All of these folders contain a script file to start the application via the command line and of course the jar-file. Other files are necessary for computing complex mappings, but not appreciable for the user. Beside the Windows and Linux version, the CD version also contains the java virtual machine and so the program can be executed on a windows computer where java is not installed. Both other folders, *Complex_Mapping_Windows* and *Complex_Mapping_Linux*, can be copied to the computer and use the installed java virtual machine if the environment variable is correctly set.
To start the application, open a shell and switch to the directory where the folder, containing the starting script, is located. Execute the script file whereby three or four parameters have to be passed.
Windows:
D:\Complex_Mapping_Windows>run.bat *ontology1 ontology2 referenceAlignment* [*fileToWrite*].
Linux:
x@y: /D/Complex_Mapping_Linux>./run.sh *ontology1 ontology2 referenceAlignment* [*fileToWrite*].

The first two parameters *ontology1* and *ontology2* are the paths to the ontologies of these the complex mapping should be created, the third one *referenceAlignment* the path to the reference alignment. All three parameters are necessary and cannot be omitted. To get a file containing the complex correspondences and not only to see them on command line, *fileToWrite* is optional and specify the path to a file where the correspondences should be written to. Only the URIs in shortened form are outputted for a better readability.
An example for Windows: D:\Complex_Mapping_Windows>run.bat cmt.owl ekaw.owl cmt-ekaw.rdf
Entering this command, the complex correspondences between the ontologies cmt and ekaw will be computed and printed, if files cmt.owl, ekaw.owl and cmt-ekaw.rdf are located in the same folder as run.bat.

Computing the complex alignment, especially the similarity between names, is based on the mid thresholds described in Sectionsec:setting and sec:results because they seemed to be the best.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 28.05.2009 Unterschrift