On the retrieval of ontology-based annotated multimedia by the use of instance matching techniques given multiple ontology versions

submitted by
Gerret Brandt

supervised by
Prof. Dr. Ralf Möller
M.Sc. Irma Sofia Espinosa Peraldi

Hamburg University of Science and Technology
Software Systems Institute (STS)

Declaration

I declare that:
this work has been prepared by myself,
all literal or content based quotations are clearly pointed out,
and no other sources or aids than the declared ones have been used.

Hamburg, 08th November 2008
Gerret Brandt

Contents

Chapter 1

Introduction

The increase of multimedia content in databases, filesystems, and on the Web has generated a need for an effective search and retrieval of multimedia objects. Prerequisite for an effective retrieval is the separation of semantics from data and a representation of the semantics in a machine readable format. Ontologies are a means to realize this separation and have recently seen an increase in reasearch due to the idea of the Semantic Web.

Information can be retrieved from ontologies by employing query languages like SPARQL[1] or nRQL[2]. These query languages have a formal syntax and semantics and return only exact matches. To construct queries in these languages, detailed knowledge about the ontology, like the names of concepts and roles and the relationship between them, is necessary. Furthermore, the semantics of the ontology concepts and roles have to be known.

This project work examines a query by example approach to information retrieval. The idea is to specify an instance representing a real world object, for example, an image of a highjump event, and using an instance matching algorithm to retrieve similar instances, i.e., similar images of highjump events.

The increase in the number of ontologies has recently lead to an increase in the reasearch of ontology matching. Most of these matching approaches focus on finding mappings of concepts and roles and up to now the attention on instances for the purpose of ontology matching has been poor [3]. Nevertheless, two promissing algorithms for instance matching, namely HMatch 2.0 and FOAM, are evaluated in this project work for the purpose of multimedia retrieval.

[1] http://www.w3.org/TR/rdf-sparql-query/

[2] http://www.racer-systems.com/products/racerpro/index.phtml

Chapter 2

Ontology Matching

The ontology matching problem is the problem of finding semantic mappings between entities of two given ontologies. Mappings consist of two entities with a similar intended meaning and a similarity value specifying the degree of similarity [7]. Entities are concepts, roles or instances. This chapter concentrates on mappings for concepts and roles. Instance matching and the resulting instance mappings are discussed in the next chapter.

Mappings between entities can be one-to-one or one-to-many. The first kind of mapping returns only the best match, while the second kind of mapping acknowledges the fact that an entity can be similar to more than one other entity. This is especially useful for information retrieval where not an exact match is desired, but instead a set of similar instances is expected.

There are many options to establish mappings between between concepts and roles. These can be classified as [10]:

Terminological Comparison compares entity names. The name of an entity can be retrieved from a label or the resource identifier.

Internal Structure Comparison compares the internal structure of entities, i.e., the value range or cardinality of properties.

External Structure Comparison compares entities by considering the relationships with other entities.

Extension Comparison compares the known extensions of entities (in general instances of concepts).

All four methods can be combined to establish a mapping and evaluate a similarity value. The terminological comparison is the only method that does not require an initial mapping, all other methods need precalculated similarity values for roles or concepts.

2.1 Terminological Comparison

Two approaches to terminological comparison exist: a semantic approach and a syntactic approach [4]. The semantic approach interprets names as terms and looks them up in a thesaurus to find relationships between them. Common terminological relationships are synonymy, meronymy, hyponymy, and hypernymy.

Synonymy Synonyms are different words with identical or at least similar meanings and the state of being a synonym is called synonymy.

Meronymy A meronym means part of a whole. For example, leaf is a meronym of tree because a leaf is part of a tree.

Hyponymy Hyponymy is the relationship between a general term and a specific instance. Hyponyms are a set of related words whose meaning are specific instances of a more general word. For example, red, white and blue are hyponyms of colour.

Hypernymy Hypernymy is the oposite of Hyponymy

Each terminological relationship is associated with a weight, according to the degree of similarity it conveys. The similarity value is evaluated as the strength of the highest-strength path of terminological relationships between terms, or 0 if no path exists. The strength of a path is the product of the weights of each relationship on the path.

The syntactic approach interprets names as strings of characters and evaluates a similarity measure based on string distance. Common string distance measures are edit distance, n-grams, soundex, etc.

2.2 Internal Structure Comparison

The matching of entities based on the internal structure, like the value range of properties or the cardinality of properties, is not implemented by the matching algorithms examined in this students project

2.3 External Structure Comparison

The external structure comparison compares the contexts of two entities. An entity context consists of relationships of an entity with other entities. Relationships can be properties, property axioms, or class axioms, depending on the kind of entity. The basic idea is, that entities that partcipate in similar relationships are similar to each other. Basically, entities are interpreted as the root element of a tree structure consisting of relationships as edges and entities as nodes. Two tree structures are compared by comparing the corresponding relationships and entities. The evaluated similarity values of the

nodes are propagated to the root elements. Therefore, basic similarity values for relationships and entities have to be calculated before an external structure comparison can be performed.

2.4 Extension Comparison

The extension comparison interprets concepts as sets of instances. If the intersection of two sets of instances is non-empty, the corresponding concepts must be similar. The more the instance sets overlap, the more similar are the concepts [6]. The same basic idea can be applied to properties and property instances. Extension comparison is not implemented by the matching algorithms examined in this project work.

2.5 Applications

Ontology Mappings can be utilized to support the following tasks:

Ontology Merging is the creation of a new ontology from one or more source ontologies [5].

Ontology Enrichment is the process of extending an ontology, through the addition of new concepts, roles and rules into an ontology [1].

Ontology Population is the process of inserting concept instances and role instances into an existing ontology [3].

Chapter 3

Instance Matching

Instance matching is a subtask of ontology matching and refers to the finding of mappings between instances. Basically, there are three different methods to establish a mapping between instances and each method considers a different set of instance features:

Terminological Comparison compares the instance names. The name of an instance can be retrieved from a label or the resource identifier.

Internal Structure Comparison compares the datatype properties and datatype property values of instances

External Structure Comparison compares instances by considering the relationships with other instances, i.e., object properties and object property fillers.

Type Comparison compares the direct classes of instances and their superclasses.

All four methods can be combined to establish a mapping and evaluate a similarity value. The terminological comparison is the only method that does not require an initial mapping, all other methods need precalculated similarity values for properties or classes.

Instance matching can serve two different goals:

Identity Matching aims to produce mappings between entities that refer to the same real-world objects.

Similarity Matching aims to produce mappings between entities that refer to similar real-world objects.

Identity matching is realised by considering only those instance features, that are relevant for identification or by emphasizing these features when evaluating the similarity value. In these cases the similarity values denote a degree of identity [3].

3.1 Terminological Comparison

Terminological comparison for instances is identical to terminological comparsion for concepts and roles described in Chapter 2. The evaluation of a similarity values for instances based on instance names is only sensible if the instances have meaningful names. This may not be the case if the ontology is populated by an automatic process.

3.2 Internal Structure Comparison

Features that are especially suited for identity matching are datatype properties and their values, i.e., the internal structure. Not all datatype properties are equally suited, and it is not possible to generally state which kind of datatype properties are more adequate for identity matching. Therefore, in most cases the relevant properties have to be configured manually before the matching process is started. For example, the date of birth of a person is certainly more relevant for identification than her eye color.

On the other hand, datatype properties or often not useable for similarity matching because they frequently convey a degree of identity and therefore emphasize the differences. Here is an example from the athletics domain: If images of 100m sprints are associated with the corresponding winner time, these time values differentiate images of different 100m events from each other. It depends on the use case if this is desired. If one wants to find similar images regardless of the actual event depicted, the influence of the winner times on the similarity value calculation should be decreased. To generalize, internal structure comparison, i.e. comparison of datatype properties and their values, is often more suitable for identity matching than for similarity matching when considering images.

3.3 External Structure Comparison

External structure comparison compares the context of an instance, consisting of object properties and the corresponding fillers. This kind of comparison is especially suited for a similarity matching of instances representing images. Image content is mostly modeled by instances and their object roles, representing the relationship between these instances, and to lesser extend by datatype roles and their values.

3.4 Type Comparison

Type comparison compares the sets of direct classes of instances and their superclasses to evaluate a similarity value. The basic idea is that instances can only be similar if they are members of similar classes.

Chapter 4

HMatch 2.0

HMatch 2.0 is a software tool for ontology matching and supports the matching of concepts, roles, and instances. It provides four matching components for linguistic, contextual, structural and instance matching [2]. These components are invoked sequentially. The output of each component is a set of mappings, each comprising a pair of entities and a similarity value in the range $[0, 1]$. The first component invoked has to be the linguistic or the strucural matching component, because these are the only components that do not need an initial mapping. Apart from the structural component, the components can be chosen arbitrarily in the subsequent stages and invoked multiple times.

4.1 Linguistic Matching

The linguistic matching component tries to find mappings between concepts and roles based on their names. The strenght of this mapping is expressed by a linguistic affinity value in the range $[0, 1]$.

The component can employ the WordNet thesaurus to detect a semantic relationship between names. Semantic relationships considered by HMatch are the synonym, hypernym, hyponym, and meronym relationships. Paths of semantic relationships between two names are assigned a strength value. The strength of a path is evaluated by multiplying the weights of each semantic relationship on the path. The weights are chosen from the range $[0, 1]$, such that semantic relationships that convey closer similarity between terms get assigned a higher value (see Table 4.1). The strength of the highest-strength path is chosen as the linguistic affinity value.

If HMatch fails to find a semantic relationship between the names, a syntactic matching approach is taken. In this case the names are interpreted as strings and a string similarity measure is evaluated. The default similarity measure of HMatch is called q-grams distances and is based on trigrams. Each string is converted into the set of all character sub-sequences of length 3

(trigrams). The sets are compared by counting the number of occurrences of each trigram in both sets, respectively, and subtracting these numbers. The differences are then summed up and set in relation to the total number of trigrams to obtain a similarity value in the range $[0, 1]$.

4.2 Contextual Matching

The contextual matching component calculates a semantic affinity measure SA between concepts based on the linguistic affinity LA, evaluated by the linguistic matching component, and an contextual affinity CA.

$$SA = w \cdot LA + (1 - w) \cdot CA$$

The contextual affinity CA evaluation is based on the context of both concepts. Depending on the applied matching model, the context of a concept can include semantic relations with other concepts, properties and object property fillers. HMatch provides four contextual matching models [4]:

Surface Model The surface model considers only concept names. The context is empty and the semantic affinity equals the linguistic affinity $SA = LA$.

Shallow Model The shallow model considers concept names and properties.

Deep Model The deep model considers concept names, properties, and the relationships with other concepts.

Intensive Model The intensive model considers concept names, properties, relationships with other concepts, and object property fillers.

The shallow model is an extension of the surface model, the deep model is an extension of the shallow model, and the intensive model is an extension fo the deep model. The semantic relations between concepts considered for contextual matching are the sub-concept and the super-concept relationships. The set of considered object property fillers is constrained to the fillers of object properties participating in an existential restriction.

Semantic Relationship	Weight
Synonym	1.0
Hypernym	0.8
Hyponym	0.8
Meronym	0.5

Table 4.1: Weights of Semantic Relationships

In the intensive model the contextual affinity CA of two concepts is calculated by evaluating a similarity value for four pair of sets, namely the pair of sub-concept sets, the pair of super-concept sets, the pair of property sets, and the pair of property filler sets. The calculation of the set similarity values depends on the matching strategy:

Standard Strategy This is a non-symmetric strategy. For each element in set 1 the best matching element in set 2 is searched by retrieving the similarity values from an input mapping. The best similarity values for each element in set 1 are summed up and divided through the total number of elements in set 1.

Dice Coefficient Strategy The dice coefficient strategy increases a counter for each element in set 1 for that a mapping to an element in set 2 exists. At last, the counter value is doubled and divided by the total number of elements in both sets.

Both strategies require an input mapping because the evaluation of the set similarity values depends on existing similarity values between the individual set elements. The semantic affinity is produced by calculating the average of the four set similarty values.

4.3 Structural Matching

The structural matching component works on the RDF graph structure of the ontologies and can also define mappings for elements without a name, such as anonymous classes. It does not take any mappings as input and therefore has to be invoked first. If two ontologies do not provide sufficient linguistic information, this component can replace the linguistic matching component.

4.4 Instance Matching

The goal of the instance matching component is to determine instances that represent the same real world object [2], i.e., identity matching.

Prerequisite for a mapping between two instances is the existence of a mapping between their direct classes or superclasses of these classes. If there exists a mapping between two classes, then HMatch evaluates a similarity value between all of their instances. The similarity measure is based on instance roles and role fillers. An instance with its roles and role fillers is seen as a tree structure with the instance as root element. To calculate the similarity value of two instances, the tree structures of these instances are traversed along matching roles and role fillers. The mappings corresponding similarity values of roles are retrieved from an input mapping. When the algorithm reaches a leaf, i.e., a datatype role filler, a similarity value is calculated from the datatype values. This similarity value is propagated to the instances of the higher level and from there it is propagated further up till it reaches the

root element This means that the instance matching process is solely based on similarity of datatype values.

A datatype matcher can be assigned to each datatype property. Table 4.2 lists the matchers available for each datatype. If no value matcher was assigned to a datatype property, the default string matcher is applied. To specifiy the relevance of a datatype property for identity matching, a weight can be applied to each datatype property.

Datatype	Matchers
Number	AgeDifference, PerformanceDifference, TimePerformanceDifference
String	Default, GeographicAffinity
Date	DateDifference

Table 4.2: Datatype Value Matchers

Chapter 5

FOAM/NOM

FOAM[8] is a software tool to align OWL ontologies. It features two slightly different matching algorithms, named NOM[1] and QOM[2]. QOM differs from NOM in that it restricts the inital input for the matching process to a subset of all possible entity pairs. In subsequent steps this subset is enlarged to refine the matching results. Because of this, QOM is faster than NOM, but the mapping quality is decreased [7][9]. Because of this I will use NOM in this project work.

FOAM/NOM employs an iterative approach to the alignment problem. The input to each step is the exhaustive set of entity pairs, each pair comprising entities of the two ontologies and each pair comprising only concept, role or instance entities. In each iteration, several similarity values are computed for each entity pair, for instance, one similarity value derived from the entity names and other similarity values derived from relationships with other entities. In the next step the similarty values for each entity pair are combined into an aggregated similarity value which serves as input for the next iteration. In Section 5.1 the computations of the similarity values are described. Section 5.2 describes the aggregation of the similarity values and Section 5.3 describes the iteration process.

5.1 Matching

For each entity pair several features are extracted and similarity values are calculated from these features. Which features are considered depends on the entity type (class, property or instance).

1. Features extracted from classes:

[1] Naive Ontolgoy Matching
[2] Quick Ontology Matching

- label
- resource identifier
- the set of superclasses
- the set of subclasses
- the set of datatype properties where the class is in the domain
- the set of object properties where the class is in the domain
- the set of object properties where the class is in the range

2. Features extracted from datatype properties:

   - label
   - resource identifier
   - the set of classes that are in the domain of the property
   - the set of super properties
   - the set of sub properties
   - the set of tuples comprising instances and corresponding datatype values (the extension of the propery)

3. Features extracted from object properties:

   - label
   - resource identifier
   - the set of classes that are in the domain of the property
   - the set of classes that are in the range of the property
   - the set of super properties
   - the set of sub properties
   - the set of tuples comprising instances (the extension of the property)

4. Features extracted from instances:

   - label
   - resource identifier
   - the set of classes an instance is a member of
   - the set of tuples comprising datatype properties and corresponding values
   - the set of tuples comprising object properties and corresponing instances

Since FOAM does not employ a reasoner, only the explicitely asserted axioms are used to generate the feature sets. With two exceptions, similarity values are evaluated only between identical features. The exceptions are two similarity values for class entities, which are evaluated by matching the set of subclasses with the set of superclasses and vice versa.

The two features entity label and resource identifier are the only features that can be extracted from all entity types. The similarity measure for these feeatures is based on the edit distance string measure:

$$sim_{String}(a,b) := max(0, \frac{min(|a|,|b|) - distance(a,b)}{min(|a|,|b|)})$$

The similarity measure for entity sets is evaluated by one of the following functions:

AvgMax retrieves for each entity in set 1 the greatest similarity value with any entity in set 2 and evaluates the average over all these values.

MaxMax retrieves the maximum similarity value of any pair of entities in sets 1 and 2.

The similarity measure for sets of tuples is evaluated by the following function:

TupleAvgMax retrieves for each tuple in set 1 the greatest similarity value with any tuple in set 2 and evaluates the average over all these values. The similarity value of a pair of tuples is computed by retrieving the similarity values of the first elements and the second elements, respectively, and then choosing the minimum of both values. The similarity values of tuple elements are retrieved from previous mappings if the elements are entities (concepts, roles, or instances). If the elements are datatype property values, than the edit distance measure is applied when the value is of type string, otherwise the value 0 is returned.

The result of the functions AvgMax, MaxMax, and TupleAvgMax depends on matching results of previous iterations. With each new iteration step the previously computed similarity values are propagated to neighbouring entities.

5.2 Aggregation

In this step of the iteration the similarity values of each entity pair are combined into an aggregated similarity value. Depending on the configuration of FOAM/NOM the similarity values are unchanged or transformed by a sigmoid function ($sig : [0,1] \rightarrow [0,1]$) before the aggregation process starts. The aggregated value is computed by using a weighted sum of the similarity values.

$$sim_{Agg} = \frac{\sum_{k=1}^{n} w_k \cdot sim}{\sum_{k=1}^{n} w_k}$$

The weights w_k are are not configurable and set to 1.0 for most similarity values. The weights for label and resource identifier similarity values are assigned the values 2.0 and 5.0, respectively. This increases the relevance of entity names in comparison to the other entity features. The weights for the two similarity values evaluated by matching a set of superclasses with a set of sublcasses are assigned the value -1.0 and have therefore a negative effect on the aggregated value.

5.3 Iteration

Most similarity values are extracted from entity features that correspond to relationships with other entities and therefore depend on results of previous iterations. For these features an initial similarity mapping is needed. These initial mapping is provided in the first iteration by the linguistic matching of labels and resource identifier. Therefore, in the first iteration, entities are matched only by name. In the subsequent iterations previous matching results are propagated to neighbouring entities. After an indeterminate number of iterations the similarity values will reach a steady state.

Chapter 6

Evaluation

6.1 Test Cases

This chapter evaluates the instance matching algorithms. First, several test cases are introduced and then these test cases are evaluated and the results are discussed for each matching algorithm. The test cases each consist of a TBox and two simple Aboxes, called ABox 1 and Abox 2-x, where x is the test case number. The TBox is based on the Athletics Event Ontology[1] from the BOEMIE[2] project and describes the domain of athletics. The ABoxes represent an image depicting an athletics event like high jump or pole vault.

ABox 1 is the same for all test cases and represents a specific high jump event. The ABox is depicted in Figure 6.2, and the image represented by ABox 1 is shown in Figure 6.1. For each test case, ABox 1 is matched against an ABox representing a similar event. Each ABox has a different XML base address, ensuring that no pair of resource identifiers of two different ABoxes are identical. Matching instances with the same resource identifier would be trivial.

Ontologies evolve over time. New concepts, roles and instances are introduced and concepts and roles are renamed or changed otherwise. ABoxes built against the older TBox version have to be matched against the newer one so that the information does not get lost. Therefore, in some test cases the ABoxes import different TBox versions to evaluate the effects of changes to the TBox.

[1] http://repository.boemie.org/ontology_repository_tbox/aeo.owl
[2] http://www.boemie.org

Figure 6.1: Abox 1 Image

```
...
<rdf:RDF
   xmlns="http://www.boemie.org/ontologies/abox1.owl#"
   xml:base="http://www.boemie.org/ontologies/abox1.owl"
>

...

<aeo:HighJump rdf:ID="newInd_02">
   <aeo:hasParticipant rdf:resource="#newInd_01"/>
   <aeo:hasPart rdf:resource="#mlc03"/>
</aeo:HighJump>

<aeo:HorizontalBar rdf:ID="mlc03"/>

<aeo:Person rdf:ID="newInd_01">
   <aeo:hasPart rdf:resource="#mlc02"/>
   <aeo:hasPart rdf:resource="#mlc01"/>
</aeo:Person>

<aeo:PersonBody rdf:ID="#mlc01">
   <aeo:isBehind rdf:resource="#mlc02"/>
   <aeo:isAtBothSides rdf:resource="#mlc03"/>
</aeo:PersonBody>

<aeo:PersonFace rdf:ID="mlc02">
   <aeo:isBelow rdf:resource="#mlc03"/>
</aeo:PersonFace>
...
```

Figure 6.2: Abox 1

6.1.1 Test Case 1

The first test case matches ABox 1 against Abox 2-1. Both ABoxes represent the same image and have the same internal structure. Even the instance names, represented by the fragment identifier of the resource identifiers, are identical. The result of this test case can be compared to the result of Test Case 2 to asses the relevance of instance names for the matching result. This is the only test case where the instance names match between the two ABoxes.

Figure 6.3: ABox 2-1 Image

```
...
<rdf:RDF
   xmlns="http://www.boemie.org/ontologies/abox2-1.owl#"
   xml:base="http://www.boemie.org/ontologies/abox2-1.owl"
>

<aeo:HighJump rdf:ID="newInd_02">
   <aeo:hasParticipant rdf:resource="#newInd_01"/>
   <aeo:hasPart rdf:resource="#mlc03"/>
</aeo:HighJump>

<aeo:HorizontalBar rdf:ID="mlc03"/>

<aeo:Person rdf:ID="newInd_01">
   <aeo:hasPart rdf:resource="#mlc02"/>
   <aeo:hasPart rdf:resource="#mlc01"/>
</aeo:Person>

<aeo:PersonBody rdf:ID="#mlc01">
   <aeo:isBehind rdf:resource="#mlc02"/>
   <aeo:isAtBothSides rdf:resource="#mlc03"/>
</aeo:PersonBody>

<aeo:PersonFace rdf:ID="mlc02">
   <aeo:isBelow rdf:resource="#mlc03"/>
</aeo:PersonFace>
...
```

Figure 6.4: Abox 2-1

6.1.2 Test Case 2

The second test case is similar to the first test case, in that the ABoxes represent both the same image and have the same internal structure. But this time the instance names (i.e. the fragment identifiers) are different. By comparing the result of Test Case 2 to the result of Test Case 1, one can assess the relevance of instance names for the matching process.

Figure 6.5: ABox 2-2 Image

```
...
<aeo:HighJump rdf:ID="xyz12">
   <aeo:hasParticipant rdf:resource="#xyz11"/>
   <aeo:hasPart rdf:resource="#abc13"/>
</aeo:HighJump>

<aeo:HorizontalBar rdf:ID="abc13"/>

<aeo:Person rdf:ID="xyz11">
   <aeo:hasPart rdf:resource="#abc12"/>
   <aeo:hasPart rdf:resource="#abc11"/>
</aeo:Person>

<aeo:PersonBody rdf:ID="abc11">
   <aeo:isBehind rdf:resource="#abc12"/>
   <aeo:isAtBothSides rdf:resource="#abc13"/>
</aeo:PersonBody>

<aeo:PersonFace rdf:ID="abc12">
   <aeo:isBelow rdf:resource="#abc13"/>
</aeo:PersonFace>
...
```

Figure 6.6: Abox 2-2

6.1.3 Test Case 3

The third test case matches ABoxes representing images of different high jump events and therefore having different ABox structures. In the image represented by ABox 2-3 (Figure 6.8) and shown in Figure 6.7 the athletes body and face are both above the horizontal bar. In the image corresponding to Abox 1 (Figure 6.1) the athletes face is below and the body is before the horizonal bar.

Figure 6.7: ABox 2-3 Image

```
...
<aeo:HighJump rdf:ID="xyz12">
   <aeo:hasPart rdf:resource="#abc13"/>
   <aeo:hasParticipant rdf:resource="#xyz11"/>
</aeo:HighJump>

<aeo:HorizontalBar rdf:ID="abc13"/>

<aeo:Person rdf:ID="xyz11">
   <aeo:hasPart rdf:resource="#abc12"/>
   <aeo:hasPart rdf:resource="#abc11"/>
</aeo:Person>

<aeo:PersonFace rdf:ID="abc12">
   <aeo:isAbove rdf:resource="#abc13"/>
</aeo:PersonFace>

<aeo:PersonBody rdf:ID="abc11">
   <aeo:isBehind rdf:resource="#abc12"/>
   <aeo:isAbove rdf:resource="#abc3"/>
</aeo:PersonBody>
...
```

Figure 6.8: Abox 2-3

6.1.4 Test Case 4

In this test case 4 the differences between the ABoxes are even more increased. ABox 1 is matched against ABox 2-4 (Figure 6.10) representing a pole vault event (Figure 6.9). The concepts PoleVaullt and HighJump are disjoint, but have the same super concept. In both images the same kind of conceps are present, namely a Person with a PersonBody and a PersonFace, and a HorizontalBar.

Figure 6.9: ABox 2-4 Image

```
...
<aeo:PoleVault rdf:ID="xyz21">
   <aeo:hasPart rdf:resource="#abc51"/>
   <aeo:hasPart rdf:resource="#abc41"/>
   <aeo:hasParticipant rdf:resource="#xyz11"/>
</aeo:PoleVault>

<aeo:Pole rdf:ID="abc41"/>

<aeo:Pillar rdf:ID="abc51"/>

<aeo:Person rdf:ID="xyz11">
   <aeo:hasPart rdf:resource="#abc21"/>
   <aeo:hasPart rdf:resource="#abc11"/>
</aeo:Person>

<aeo:PersonFace rdf:ID="abc21">
   <aeo:isBelow rdf:resource="#abc31"/>
</aeo:PersonFace>

<aeo:PersonBody rdf:ID="abc11">
   <aeo:isBehind rdf:resource="#abc21"/>
   <aeo:isAtBothSides rdf:resource="#abc31"/>
</aeo:PersonBody>

<aeo:HorizontalBar rdf:ID="abc31"/>
...
```

Figure 6.10: Abox 2-4

6.1.5 Test Case 5

For this test case the effects of different TBox versions are evaluated. Specifically, in the TBox corresponding Abox 2-5 (Figure 6.12) the concept HighJump is renamed to HochSprung. Besides this change, the Aboxes are the same as in Test Case 2, i.e., the two Aboxes represent the same image (Figure 6.11).

Figure 6.11: ABox 2-5 Image

```
...
<aeo:HochSprung rdf:ID="xyz12">
<aeo:hasParticipant rdf:resource="#xyz11"/>
<aeo:hasPart rdf:resource="#abc13"/>
</aeo:HochSprung>

<aeo:HorizontalBar rdf:ID="abc13"/>

<aeo:Person rdf:ID="xyz11">
<aeo:hasPart rdf:resource="#abc12"/>
<aeo:hasPart rdf:resource="#abc11"/>
</aeo:Person>

<aeo:PersonBody rdf:ID="abc11">
<aeo:isBehind rdf:resource="#abc12"/>
<aeo:isAtBothSides rdf:resource="#abc13"/>
</aeo:PersonBody>

<aeo:PersonFace rdf:ID="abc12">
<aeo:isBelow rdf:resource="#abc13"/>
</aeo:PersonFace>
...
```

Figure 6.12: Abox 2-5

6.1.6 Test Case 6

The last test case evaluates the effects of an equivalent class axiom. In a new version of the TBox the concept HochSprung is introduced in an equivalent class axiom (Figure 6.15). This axiom specifies, that all instances of HochSprung are also instances of HighJump and vice versa. ABox2-6 is the same as Abox2-5 (Figure 6.12)

Figure 6.13: ABox 2-6 Image

```
<aeo:HochSprung rdf:ID="xyz12">
   <aeo:hasParticipant rdf:resource="#xyz11"/>
   <aeo:hasPart rdf:resource="#abc13"/>
</aeo:HochSprung>

<aeo:HorizontalBar rdf:ID="abc13"/>

<aeo:Person rdf:ID="xyz11">
   <aeo:hasPart rdf:resource="#abc12"/>
   <aeo:hasPart rdf:resource="#abc11"/>
</aeo:Person>

<aeo:PersonBody rdf:ID="abc11">
   <aeo:isBehind rdf:resource="#abc12"/>
   <aeo:isAtBothSides rdf:resource="#abc13"/>
</aeo:PersonBody>

<aeo:PersonFace rdf:ID="abc12">
   <aeo:isBelow rdf:resource="#abc13"/>
</aeo:PersonFace>
```

Figure 6.14: Abox 2-6

```
...
<owl:Class rdf:ID="HochSprung">
   <owl:equivalentClass rdf:resource="#HighJump"/>
</owl:Class>
...
```

Figure 6.15: TBox 2-6

6.2 HMatch 2.0

In this project work the retrieval of multimedia objects with instance matching techniques is tested on the Athletics Event Ontology from the BOEMIE project and the test scenario is limited to image retrieval. Since the ABoxes representing the image content do not comprise datatype properties, the HMatch 2.0 algorithm can not be applied. The HMatch 2.0 version that is at present publicly available relies solely on datatype values to retrieve mappings. Instances that don't have any properties cannot be matched with any other instances by HMatch.

6.3 FOAM

FOAM/NOM is configured with the following parameters:

```
internaltoo = INTERNAL;
efficientAgenda = COMPLETE;
cutoffvalue = 0.001;
maxiterations = 13;
strategy = MANUALWEIGHTED;
removeDoubles = ALLOWDOUBLES;
scenario = NOSCENARIO;
```

6.3.1 Test Cases 1 and 2

In the first two test cases we match ABoxes representing the same image. Each ABox has a different XML base address ensuring that no pair of resource identifiers from two different ABoxes is identical (matching instances with the same resource identifier is trivial). In Test Case 1 the resource identifiers of instances describing the same real world object have the same fragment identifier, in Test Case 2 the fragment identifiers differ. Since fragment identifiers are interpreted as entity names by FOAM/NOM, they are used for linguistic matching of entities. The comparison of the first two test case results can shed light on the impact of linguistic matching on the whole matching process.

	HighJump
Test Case 1	1.000
Test Case 2	0.204

Table 6.1: Results of Test Cases 1 and 2

The results show, that the names have a significant effect on the matching process. Although in Test Case 2 only the instance names were changed, the similarity value has dropped nearly 80 percent. Since the names of instance are seldom meaningful (unlike concept and role names), linguistic matching for instances is in most cases not appropriate.

6.3.2 Test Cases 3 and 4

	HighJump
HighJump	0.201

Table 6.2: Result of Test Case 3

	PoleVault
HighJump	0.086

Table 6.3: Result of Test Case 4

The result of Test Case 3 is very similar to the result in Test Case 2 where two nearly identical ABoxes were matched. This shows that a change of relationships between instances - in this case the relationships between PersonBody and PersonFace to HorizonzalBar - doesn't significantly effect the matching result.

The result of Test Case 4 shows that the matching result is significanly decreased if the instances are of different types even if they have the same super class and a similar strucure. The reason may be that FOAM/NOM emphasizes linguistic matching over contextual matching.

6.3.3 Test Cases 5 and 6

	HochSprung
HighJump	0.122

Table 6.4: Result of Test Case 5

	HochSprung
HighJump	0.064

Table 6.5: Result of Test Case 6

Test Case 5 is the first test case where one ABox imports a different version of the TBox. In the new TBox version the concept name HighJump is replaced by HochSprung. The result can be compared with the result of Test Case 2, since the only diffence between the two test cases is the renaming of the HighJump concept. The effect of the renaming is a decrease in the matching value of 40 percent. This again shows that concept names are used as a prominent feature for instances matching by FOAM/NOM.

In Test Case 6 an additional concept HochSprung is introduced into a TBox version and this concept made equivalent to the concept HighJump. The result shows that equivalent class axioms are not considered. Otherwise, the matching result would have been the same as the matching result in Test Case 2. The result is even worse than the result of Test Case 5, because in this Test Case the concept HochSprung has no super classes, sub classes or property restrictions that could have been compared to the super classes, subclasses and property restriction of the HighJump concept. Because FOAM/NOM does not recognize a context for the concept HochSprung, the matching result is significantly diminished.

Chapter 7

Summary

This project work has shown that retrieval of multimedia using instance matching is a viable retrieval approach. Especially FOAM/NOM is able to retrieve similar images if the concept names and role names of the instances are similar. But the emphasis on the linguistic matching component has the disadvantage that concept name changes due to a new ontology version decrease the matching result significantly. The matching results were also degrade by the appliance of linguistic matching on instances names, because automatically generated instance names hold no meaningul information that can be exploited. No axioms besides the subclass and subproperty axioms are evaluated to support the matching process. HMatch 2.0 could not be evaluated, because in the examined scenario of images from the athletics domain no datatype properties were present in the ABoxes.

In some cases the consideration of additional axioms would have been beneficial and FOAM/NOM could have produced considerably better results if instance names would have been exluded from linguistic matching.

Bibliography

[1] S. Castano, S. Espinosa, A. Ferrara, V. Karkaletsis, A. Kaya, S. Melzer, R. M"oller, S. Montanelli, and G. Petasis, *Ontology dynamics with multimedia information: The boemie evolution methodology*, 2007.

[2] S. Castano, A. Ferrara, D. Lorusso, and S. Montanelli, *The hmatch 2.0 suite for ontology matchmaking*, Semantic web applications and perspectives - proceedings of the 4th italian semantic web workshop, 2007.

[3] S. Castano, A. Ferrara, S. Montanelli, and D. Lorusso, *Instance matching for ontology population*, Sebd, July 30, 2008, pp. 121–132.

[4] S. Castano, A. Ferrara, S. Montanelli, and G. Racca, *From surface to intensive matching of semantic web ontologies*, Dexa '04: Proceedings of the database and expert systems applications, 15th international workshop, 2004, pp. 140–144.

[5] J. Davies, R. Studer, and P. Warren, *Semantic web technologies - trends and research in ontology-based systems*, John Wiley & Sons, Ltd, 2006.

[6] Anhai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy, *Learning to match ontologies on the semantic web*, 2003.

[7] M. Ehrig and S. Staab, *Qom - quick ontology mapping*, Proceedings of the third international semantic web conference, 2004, pp. 683696.

[8] M. Ehrig, S. Staab, and Y. Sure, *Framework for ontology alignment and mapping.*

[9] M. Ehrig and Y. Sure, *Ontology mapping - an integrated approach*, Proceedings of the first european semantic web symposium, 2004, pp. 76–91.

[10] Jérôme Euzenat and Petko Valtchev, *Similarity-based ontology alignment in OWL-Lite*, 2004.

List of Figures