# Matching natural language activity descriptions by using self-expanding ontologies

An application to support mobility of elderly people

Bachelorarbeit

im Studiengang Wirtschaftsinformatik
der Fakultät Wirtschaftsinformatik
und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Verfasser:

Lukas Berle
(Matrikelnummer: 1565183)

Gutachter: Prof. Dr. Ute Schmid

Bamberg, 15.09.2012

**Abstract**

The aim of this work is to present and implement an approach for a self-learning system. This approach will be part of a matchmaker. The matchmaker will match elderly persons with persons who have the same interests and elderly persons with volunteers. The part of the matchmaker that is developed in this work shall assign a natural language activity description to classes in an ontology. These classes represent activities. Furthermore the approach calculates a probability from the system's view that the natural language activity description belongs to a certain class. In addition, the approach makes use of similarities between activities. This means that these approach is also able to return activities that are similar to the desired activities.

The approach is completely implemented in java and can be tested. But there are still some problems, that negatively impact the matching quality. Some possible solutions for these problems are presented in this work. If these problems will be solved or alleviated then the matchmaker will do a good work, also on long-term.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# 1 Introduction

Given a simple sentence, can a human who knows the text's language find out what is the text about? - Yes, according to the ability to read and understand, most humans will be able to say without detailed or complete reading what the text is about. This work will address the question how a software system can also find out the core of a text. The work is a part of a project that is supported by the German's Federal Ministry of Education and Research.
Germany has a disproportionate number of elderly people. Many of these people have problems with their social isolation and their lack of mobility. The question arises, how people who have the same hobbys can become acquainted with each other. Another question is, how elderly people can find people who help them doing every day activities. A final question is how it could be possible to improve the mobility of elderly people.
The aim is to create a matchmaker that helps elderly persons to request help or search partners for certain activities. Elderly persons will enter their request textual in natural language to the system. The scope of this work is to develop and examine an approach that finds out which activity an elderly person wants to do and which other activities this person could be interested in. Based on this, there are displayed people who plan to do the same or similar activities.
This thesis is structured as follows: At first, matchmaking will be defined and it is described in which domains matchmaking is used. In addition some matchmaking approaches are analyzed in order to find out if they could be used for the desired matchmaker. In chapter 3 the chosen approach, its domain and requirements for matching the natural language activity description to an ontology is described. In chapter 4 there are described some variations and problems of this approach and how the problems can be solved. Finally, in the last chapter, the results are summarized and suggestions for the future work are provided.

# 2 Matchmaking

## 2.1 Definition

There are many definitions of matchmaking. A popular definition by Lei Li and Ian reads as follows:

**Definition 1 (Matchmaking)** *Matchmaking is defined as a process that requires a repository host to take a query or an advertisement as input, and to return all the advertisements that may satisfy the requirements specified in the input query or advertisement (Li & Horrocks, 2004).*

The host that is mentioned in the definition is a matchmaking system or a matchmaker. It supports persons to find other persons for a special purpose. The matchmaker takes up the role as intermediary between the matching partners. It can match a person offering something and a person seeking for something but also match two searching persons. In the definition is also mentioned that the matchmaker returns the results that *may* satisfy the requirements. This implies that a matchmaker does not return only perfect-fitting results, but also similar results which may fit. Figure 1 shows the tasks of a matchmaker simplified.

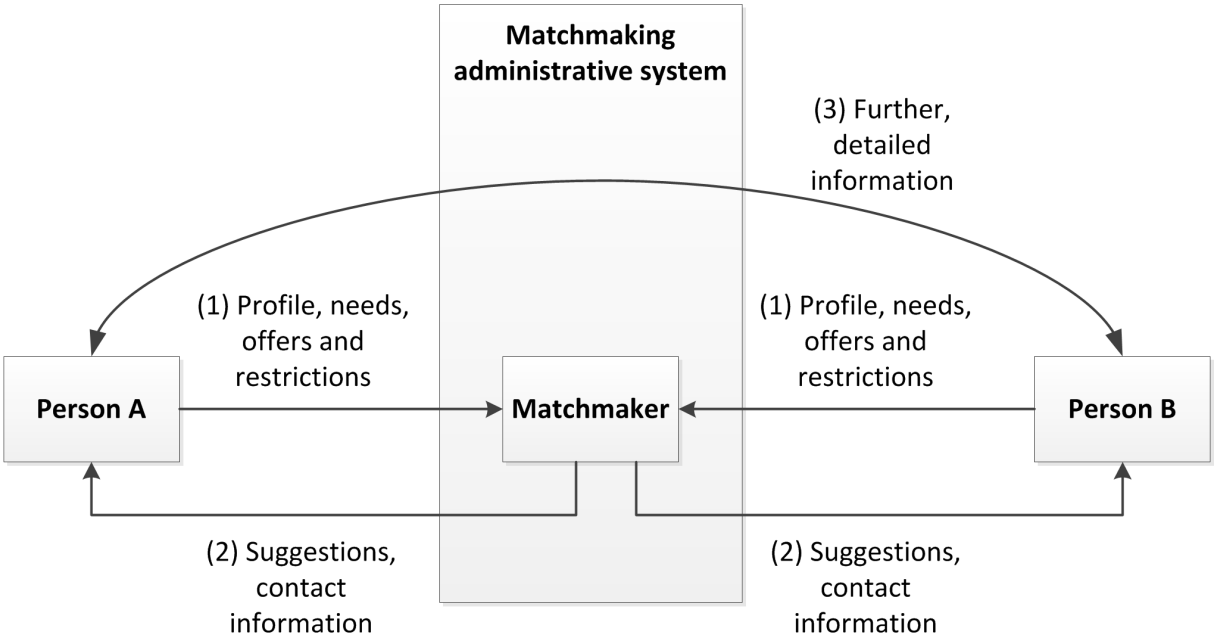



Figure 1: Tasks and position of a matchmaker

## 2.2 Matchmaking examples

Matchmakers are used in a large variety of areas. In the following list there is presented a brief overview over fields in which matchmakers are used.

- Web-based auction houses (e.g. Ebay[1], hood[2])
- Private classified advertisements (e.g. Kleinanzeigen[3])

[1]http://www.ebay.de
[2]http://www.hood.de
[3]http://www.kleinanzeigen.de

- Car sharing agencies (e.g. Mitfahrgelegenheit[4], Flinc[5])
- Web-based dating platforms (e.g. Elitepartner[6])
- Leisure and Travel partner agencies (e.g. Urlaubspartner[7])
- In Computer games in order to find persons of the same skill level (e.g. Warcraft 3[8])
- Business (In order to find internal or external resources)

## 2.3 Matching Approaches

The literature supposes some matching approaches. In order to assess the approaches each approach will be examined whether it supports semantic matching, bilateral matching and entering natural language activity descriptions.

**Definition 2 (Semantic Matching)** *Semantic matching means that the approach uses background knowledge to find semantically related information.*

**Definition 3 (Bilateral Matching)** *Bilateral matching means that both, the user who publishes a new activity and the user who searches for an activity can enter restrictions.*

**Definition 4 (Natural language activity description)** *A natural language activity description is a description, usually entered by a user. By entering this description the user wants to find partners for an activity, wants to offer help or wants to seek for help. The user who enters the description doesn't consider the system's functionality and formulate it like a thread title in an internet forum.*

**Definition 5 (Activity)** *In this work, activities are all help offers, help searches and leisure partner searches.*

### 2.3.1 ClassAds

Condor's ClassAds (classified advertisements) consist of key-value pairs. Both, the service-provider and the user who searches for a service, create ClassAds in order to

[4]http://www.mitfahrgelegenheit.de
[5]http://www.flinc.de
[6]http://www.elitepartner.de
[7]http://www.urlaubspanrter.net
[8]http://us.blizzard.com/en-us/games/war3/

describe themselves, e.g. their age or their gender. Furthermore ClassAds also contain the requirements and constraints for a match. It is also possible to rank offers. For example, it is possible to prefer partners that have special attributes or have a high value of a certain attribute. (Raman, 2001)
A 67-year-old female who searches a person older than 40 for a trip to the opera would create the following ClassAd if she prefers a person with old age:

[ Name = “Ingrid Oberhagen”;
Age = 67;
Gender = w;
Requirements = other.Age>40 && other.Activity == “opera”
Rank = other.Age ]

**Advantages** Condor supports bilateral matching, because both service provider and service seeker, can set constraints.
Another advantage is that ClassAds are semi-structured and so easy extendable and flexible. Thus, one can leave some attributes, e.g. “age” or “gender”. But it is also possible to add other requirements the desired partner has to fulfill. Furthermore the user can rank the potential partners by setting preferences to specific attributes. So it is possible to find partners that have an old age, that have the same hobbies or that have many persons as friends in common.

**Disadvantages** The system is only syntax-based. Thus, service provider and service seeker must use the same syntax. According to this, the system would not be able to unify a person searching for the activity “cinema” and a person searching for the activity “movie theater” although both persons search for the same activity. In order to solve this problem, all users have to agree upon a certain vocabulary. This is, considering the usability and the large variety of vocabulary, not feasible. From this follows that ClassAds also cannot deal with natural language descriptions because it is, by natura, not feasible to restrict the words a natural language description consists of.
Another problem is that ClassAds do not support semantic matching, because it cannot make use of any background knowledge. Thus, the system cannot deal with similarities. ClassAds cannot support to display users who search for the activity “culture” all offers that contain to this concept, for example theater and opera activities. Moreover the system cannot display the user similar offers. A user who searches for the activity “opera” should also be displayed activities like musicals or theaters.

### 2.3.2 R-U-In? - Exploiting Rich Presence and Converged Communications for Next-generation Activity-Oriented Social Networking

R-U-In? is an activity-oriented social network that is based on ontologies. A user can enter an activity description of the form <location, category, time>. Then the matchmaker returns semantically related tags that are related to this input. These tags get compared to other user's interests that are collected from social networks or that are manual set by the user. Finally the user receives a map that contains the location of relevant users. (Banerjee et al., 2009)

**Advantages** *R-U-In?* supports semantic matching by using an ontology (see also chapter 3.3). According to this a search for "culture" would also take in to account that an opera or a theater belongs to the concept culture.

**Disadvantages** *R-U-In?* does not support bilateral matching. *R-U-In?* allows to enter start-time and end-time of an activity, but does not allow to enter any further constraints, for example age or gender restrictions. Also it does not support entering requirements, for example that one needs help to change the train. But one have to say that it should be possible without large problems to add this kind of constraints and requirements.
Another problem for the desired application is that *R-U-In?* does not support natural language descriptions. This means that a user must explicitly enter a category when creating an activity. Furthermore the system does not make use of entered knowledge, for example that a search for "Wagner" could belong to an opera when a user before entered "Wagner opera". Furthermore the authors do not describe how the system could deal with categories that do not exist in the ontology.

### 2.3.3 Faceted Browsing

There are two paradigms of searching in the web: the navigational search and the direct search. The navigational search "uses a hierarchy structure (taxonomy) to enable users to browse the information space by iteratively narrowing the scope of their quest". (Broder, 2006). When using the direct search a user enters (often natural language) queries in a website's text box. This is the most common type of searching the web. Also the most search engines use it.
It becomes more and more popular to combine both paradigms. This is called Faceted Browsing or Faceted Search. At the moment particular auction houses or library search systems use this kind of search: The user starts with entering a direct search. Then he or she can improve the results by using the navigational search. Figure 2 shows a

faceted search that is provided by the University Library of Bamberg.

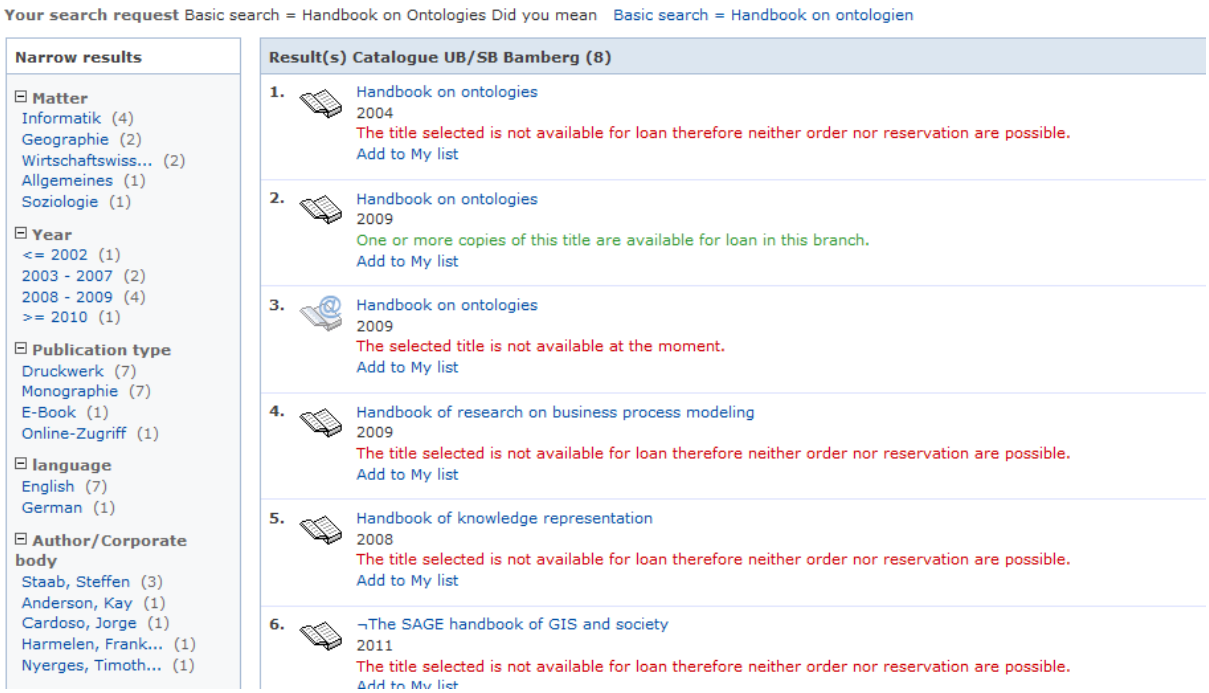



Figure 2: The search system of the University Library of Bamberg as example of a faceted search

The matchmaker can be complemented by using faceted browsing. So a user can decide for himself which kind of offers he or she prefers.

### 2.3.4 Conclusion

None of these approaches can be used to implement the desired matchmaker.Munz, Stein, Sticht, and Schmid (2012), who examined this approaches simultaneously to this work, came to the same conclusion. The objective of this work is to match a natural language activity description to an activity. The approach is presented in the next chapter. But there are also other approaches that could be used, for example by sourcing out the problem and using a search engine, for example Google[9]. It should be possible to send a natural language activity description to Google and then examine the results. So the matchmaker can get information about the context the natural language activity description belongs to. Another platform that could be used is for example Wikipedia[10]. The Wikipedia entries contain a lot of information and the entries are

[9]http://www.google.com
[10]http://www.wikipedia.org

structured hierarchically. It should be also possible to make use of this knowledge. A significant advantage of making use of the knowledge of Wikipedia is that it is possible to download it completely. In this way, the matchmaker can store a copy of Wikipedia and requests to this copy can be answered quickly when it is stored on the same server.

From all these examined approaches just faceted browsing can be used to improve the result quality of the matchmaker. But one has to make sure that elderly persons do not get confused because of a to complicated user interface.

# 3 Chosen Approach

In this chapter there is presented an approach to match the natural language activities to different activities with probabilities. Based on this approach, a matchmaker that considers restrictions and needs can be implemented. Söllner (2011) already made suggestions how these parts of the matchmaker could be implemented.

## 3.1 Domain

A large problem for elderly persons is their lack of mobility. For different reasons many elderly persons do not leave their home very often. The reason for that are for example social isolation, orientation problems, health problems or other factors (Munz et al., 2012). In order to help these persons there will be implemented a web-based matchmaker that matches persons who search for help with persons who offer help. Moreover it should match people with similar interest in order to solve the problem of social isolation. By considering other factors, like gender restriction, age restrictions or the fact that an activity is much easier to organize for both persons if they live in the same neighborhood, the user should get a ranking with the best matching potential partners.

This thesis is part of the research for this matchmaker. Persons will enter a search for help or a partner for leisure search by entering a short natural language description. The desired approach has the objective to receive this natural language activity description and to return categories the description belongs to. Furthermore it should return also categories that are similar to the desired activity. In addition to the returned categories it will return a probability that says how likely the description belongs to this category from the system's view. In later steps these categories can be used to find corresponding activities in the database. Figure 3 shows a model that describes the process of getting activity suggestions. The objective of this work is to describe, analyze and implement the parts highlighted in grey.

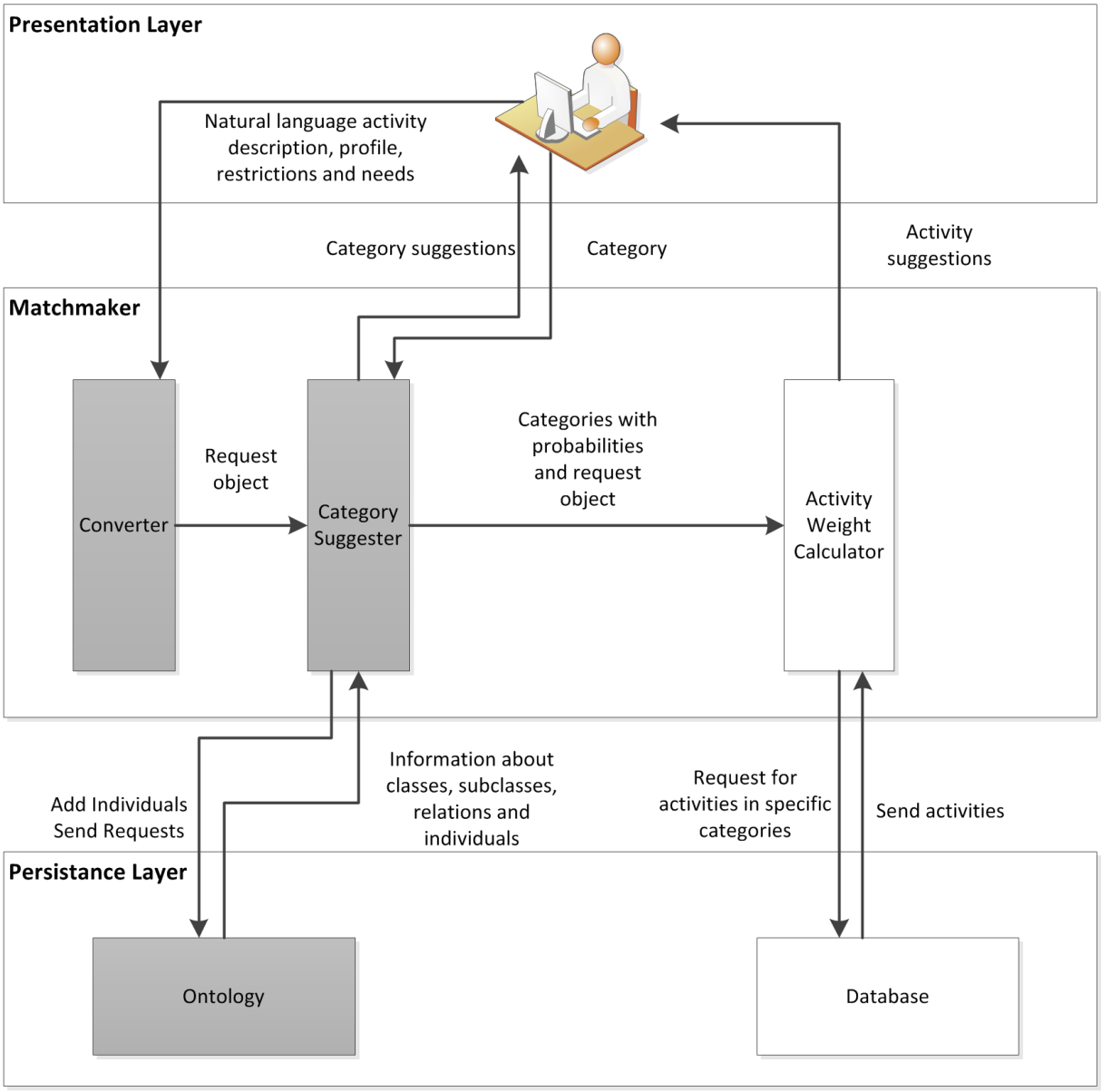



Figure 3: The process of getting activity suggestions

## 3.2 Functional Requirements

Note that there will be implemented just an approach for the category suggester. Thus the requirements relate just to the category suggester.

**Mapping Natural Language Activity Descriptions** The application must be able to map natural language activity descriptions to activities. Furthermore it must be

able to return a probability that a description belongs to an activity and return how reliable the mapping is.

**Semantic Matching** The application must be able to deal with generalizations and similarities between different kind of activities.

**Self-Learning** The application must learn from user inputs in order to improve future outputs.

## 3.3 Ontology

In 2001 the co-founder of the World Wide Web, Tim Berners-Lee and a research team presented the semantic web (Berners-Lee, Hendler, & Lassila, 2001). The core of the semantic web is to make information in the internet not just human-readable but also machine-readable. A central point of this is the use of ontologies.
Originally ontology is a term of the metaphysics, a major branch of philosophy (Staab, 2009). In Computer Science ontologies are used for modeling and representing knowledge in a certain domain. The literature supposes many different definitions of ontologies, but according Stuckenschmidt (2011), there are no definitions that are both, useful and correct, because some are to abstract and thus not useful and the others exclude special types of ontologies and are thus not correct. One abstract definition describes an ontology as follows:

**Definition 6 (Ontology)** *"An ontology is a formal, explicit specification of shared conceptualization" (Staab, 2009).*

The question arises what "conceptualization" means. Genesereth and Nilsson (1987) define conceptualization as "an abstract, simplified view of the world that we wish to represent for some purpose". Thus it is able to model a part of the world by using an ontology. In contrast, a taxonomy is just a hierarchical structure in which the objects are just related with one relation. This can be a sub-concept or a part-of relation (Daconta, Obrst, & Smith, 2003). In many ontologies there exists a taxonomy as backbone in order to create hierarchical structures (Staab, 2009). The taxonomy is a large and important part of the ontology for the desired matchmaker, too.
There are many different formal ontology languages that describe ontologies. In this work OWL2 was used. The matchmaker uses just a small part of the OWL2 features. These are classes, individuals, flags and has_Relation annotations. The following definitions may distinguish from other definitions because they are also restrictions for the way the ontology for this matchmaker must be built. If possible, the definitions contain a formal definition by using the description logic $\mathcal{ALC}$.

**Definition 7 (Class)** *A class is a collection of individuals. There are two kinds of*

*relations between classes: A Subclass relation and the inverse parent class relation. Here, a class represents an activity or a collection of activities.*

**Definition 8 (Parent Class)** *The parent class $P$ of a class $C$ is $P \mid [\, C \not\equiv P \wedge C \sqsubseteq P \wedge \neg\exists\, X \mid (X \sqsubseteq P \wedge C \sqsubseteq X)]$*

**Definition 9 (Subclass)** *The set of subclasses SC of the parent class C are all classes $S \mid (S \sqsubseteq C)$*

**Definition 10 (Direct Subclass)** *The set of direct subclasses SC of the parent class C are all classes $S \mid (parentClass(S) \equiv C)$*

**Definition 11 (Top-Level Class)** *A top-level class is a class that is a direct subclass of "Thing".*

**Definition 12 (Individual)** *An individual belongs exactly to one class. An individual is a natural language activity description that was entered by a user and was subordinated to a class. Individuals can hold a flag. The formal Definition is $a : C$*

**Definition 13 (Flag)** *A flag is a boolean value an individual can hold. If the flag is true then the individual must be further subordinated in the class hierarchy. If it is false, it was already automatically further subordinated.*

**Definition 14 (has_Relation)** *A has_Relation connection between two classes can exist in the ontology. If activities in class B and their subclasses could be relevant for a user if class A is relevant, then there should be a has_relation connection from class A to class B. See also table 1*

In figure 4 there can be seen all constructs which are used in the ontology for this matchmaker.

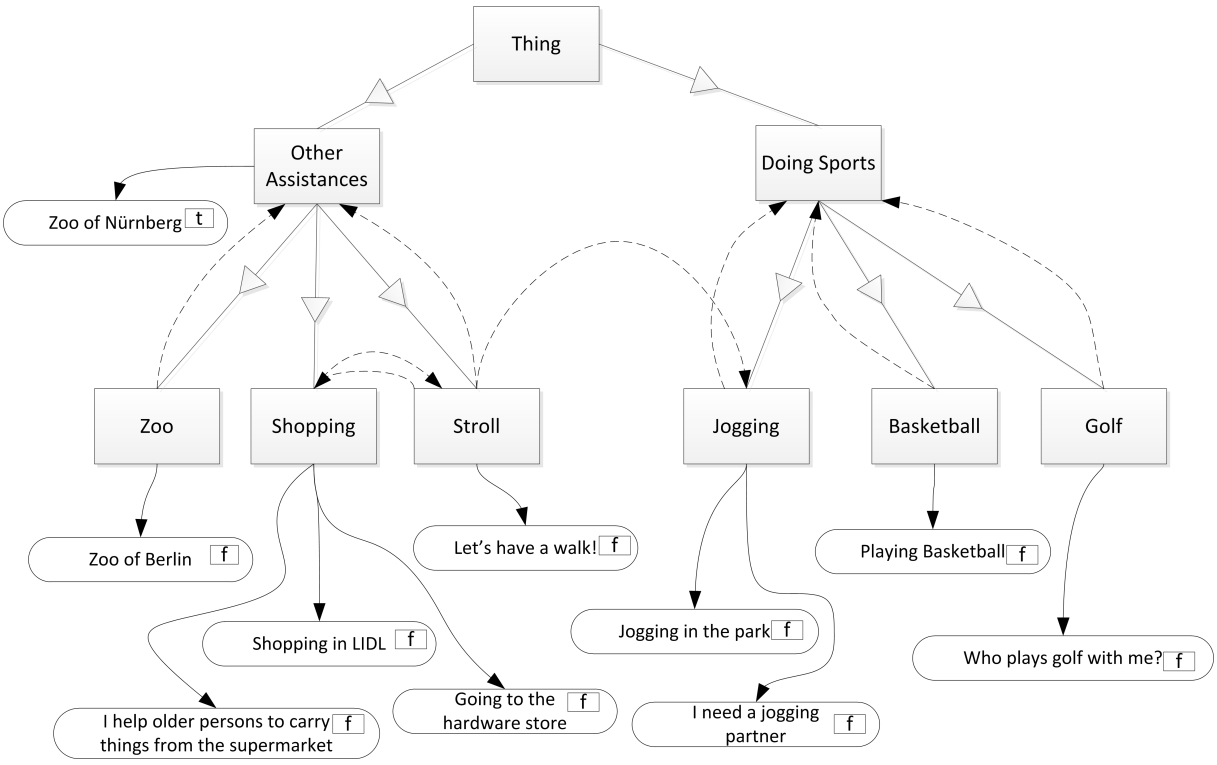



Figure 4: An exemplary ontology

In table 1 all used ontology constructs of the matchmaker are described.

| Representation | Definition | Example |
|---|---|---|
| B → C | Direct subclass | "Other Assistances" - "Zoo" |
| C → a | Individual | "Zoo" - "Zoo of Berlin" |
| C ⇢ D | has_Relation | "Stroll" - "Jogging" |
| a f | Flag | "Zoo of Berlin" - Flag: false |

Table 1: Used constructs in the matchmaker's ontology

## 3.4 Used Algorithms

### General Definitions for the Algorithm

**Definition 15 (Similarity Weight)** *A similarity weight is a global weight between 0.0 and 1.0. After each class got a weight, the weight of the classes and their subclasses that*

*are connected with a has_Relation connection from a class that have the highest weight become increased to similarity weight * highest weight.*

**Definition 16 (Strictness Factor)** *A strictness factor is a global factor between 0.0 and 1.0. A strictness factor x means that at least the 100 * x % best matching classes are returned. A high strictness factor could cause more potential matching classes than a low strictness factor.*

**Definition 17 (Quality Factor)** *A quality factor is a global factor between 0.0 and 1.0. It is used to define whether class suggestion for a specific natural language activity description is good or bad. Considering a quality factor x, then a result is good if at least x % of the suggested classes are subordinated to one top level class, otherwise the result is bad.*

### Technical Definitions

| | |
|---|---|
| C.n | If C is a reference to a class, C.n is its name |
| C.w | If C is a reference to a class, C.w is its weight |
| i.n | If i is a reference to an individual, i.n is its name |
| i.flag | If i is a reference to an individual, i.flag is its flag |
| i.class | If i is a reference to an individual, i.class is the class i is subordinated to. Formal: i.class $\equiv$ C \| [i : C $\wedge$ $\neg\exists$ X \| (X $\sqsubseteq$ C $\wedge$ i : X)] |
| equals(String s1, String s2) | Is true if s1 is exact the same String like s2 |
| clean(String s) | Returns s without stop words |
| getWords(String a) | Returns a set that contains all words of a |
| parentClass(Class C) | Returns P \| [C $\not\equiv$ P $\wedge$ C $\sqsubseteq$ P $\wedge$ $\neg\exists$ X \| (X $\sqsubseteq$ P $\wedge$ C $\sqsubseteq$ X)] |
| subClass(Class C) | Returns all S \| (parentClass(S) $\equiv$ C) |
| Class D $\equiv$ Class F | C $\sqsubseteq$ D $\wedge$ D $\sqsubseteq$ C |
| individualsOf(Class C) | Returns all i \| [i : C $\wedge$ $\neg\exists$ X \| (X $\sqsubseteq$ C $\wedge$ i : X)] |
| hasRelation(Class C) | Returns a Set of all Classes to which C is connected with a has_Relation connection |
| String s1 $\gtrless$ String s2 | Returns true if s1 is substring of s2 or if s2 is substring of s1. Otherwise it returns false |

| | |
|---|---|
| a++ | a = a + 1 |
| Set.add(c) | Adds c to the set |
| Set.remove(c) | Removes c from the set |
| Class.addInvidiual(i) | Adds individual i to class |
| Set.delete(i) | Deletes i from set |
| getTopLevelClasses | Returns a set containing all C \| C ∈ subClass("Thing") |

### 3.4.1 Weight Calculating

The algorithm calculates a normalized weight for the classes in the Ontology. The normalized weights indicate the probability that a natural language activity description belongs to a specific class.

**Algorithm 1** calculateWeights(input_a: natural language activity description)

```
1: input_o ← set of references to classes in ontology
2: input_i ← set of references to instances in ontology
3: input_k ← similarity weight
4:
5: dirctMatch = false
6: input_a = clean(input_a)
7: ForEach class ∈ input_o Do
8:   If input_a ⩾ class.n Then
9:     directMatch = true
10:    class.w = 1
11:    ∀relClass ∈ hasRelation(c): setWeightToClassAndSubClasses(relClass, input_k)
12:    If subClass(class) ≠ ∅ Then
13:      setWeightToClassAndSubClasses(class, 1)
14:    EndIf
15:  EndIf
16: EndFor
17: If directMatch = false Then
18:   ForEach word ∈ input_a Do
19:     ForEach class ∈ input_o Do
20:       ForEach individual ∈ indvidualsOf(class) Do
21:         ForEach individual_word ∈ individual.n Do
22:           If word ⩾ individual_word Then
23:             class.w++
24:           EndIf
25:         EndFor
26:       EndFor
27:     EndFor
28:   EndFor
29:   highestWeight = max_{∀classes∈input_o} (class.w)
30:   ForEach class | class ∈ input_o && class.w = highestWeight Do
31:     setWeightToClassAndSubClasses(class, highestWeight * input_k)
32:     ∀relClasses ∈ hasRelation(c): setWeightToClassAndSubClasses(relClasses, highestWeight * input_k)
33:   EndFor
34: EndIf
35: totalWeight = ∑_{class∈input_o} class.w
36: ∀class ∈ input_o: class.w = class.w / totalWeight
```

**Algorithm 2** setWeightToClassAndSubClasses(c: class, w: weight)

```
1: ForEach class ∈ {subClass(c), c} Do
2:    If class.w < w Then
3:       class.w = w
4:    EndIf
5:    If subClass(c) ≠ ∅ Then
6:       setWeightToClassAndSubClasses(c, w)
7:    EndIf
8: EndFor
```

### Direct Match

**Definition 18 (Direct Match)** *A direct match is a match that takes place if a natural language activity description is substring of a class or vice versa.*

The direct match (lines 7 - 17) in algorithm 1 is very important in the initial phase of the software usage. The reason for that is that the ontology holds very less individuals in this phase and thus the algorithm has a rather little background knowledge. The algorithm must make use of the class names in the ontology. Thus the direct match alleviate the problem of the cold start.

### Individual Based Matching

If there was not a direct match for a give natural language activity description, the matchmaker tries to find out the related category by using the individuals. Figure 5 shows an ontology with individuals.

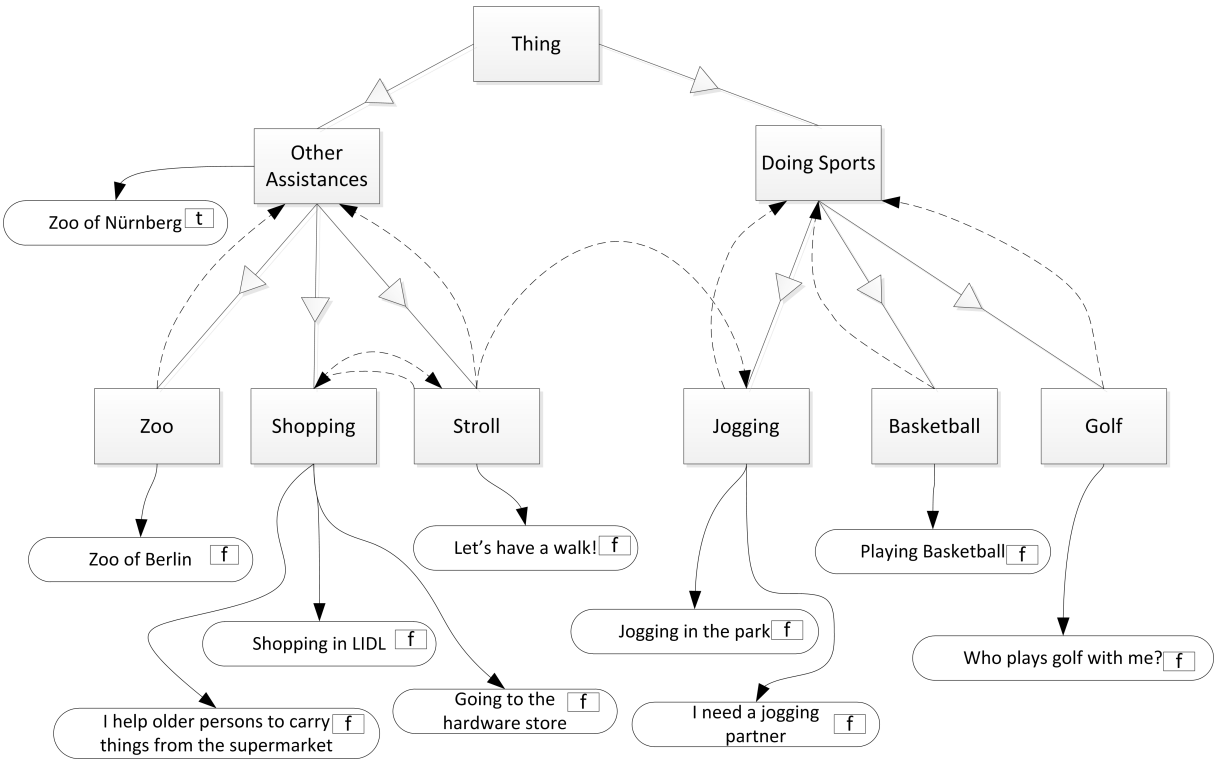



Figure 5: Ontology with individuals

Suppose that somebody enters the natural language activity description "Need help to carry drinks from LIDL". Of course the algorithm first checks if a direct match is possible. Considering the direct match definition 18, there is no direct match in the ontology in figure 5.

The next step is to start the individual based matching. At first, all stop words of the description get deleted. So the cleaned description could be "Need help carry drinks LIDL".
The next step assigns a weight to potential matching classes. If a word in the description is substring of a word in an individual or vice versa, the class weight of the individual's class get increased by one. This is done for every word in the description and for every word in the name of every individual. After doing this, all potentially relevant classes have a weight. This can be seen in figure 6. Above each individual there is an addition of 5 numbers. The first number of each addition is one if the first word of the cleaned description ("Need") is a substring of a word in the individual or vice versa, the second number is one if the second word of the cleaned description is a substring of a word in the individual or vice versa and so on. The number above each class expresses the sum of these sums.

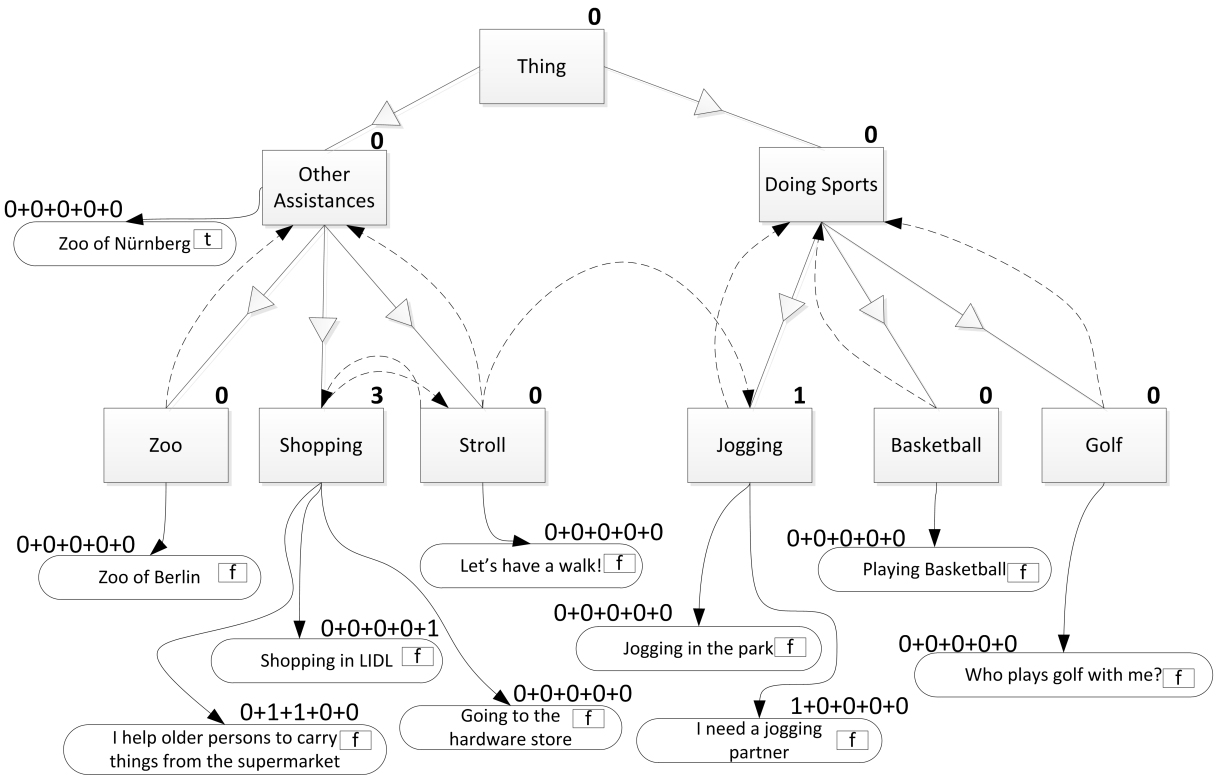



Figure 6: Ontology with weights for the natural language activity description "Need help to carry drinks from LIDL" (has_Relation connections are not considered)

In a next step, the class(es) that have the highest weight ("Shopping") are in the focus. The weight of all classes that are connected with a has_Relation connection from the highest weight classes get, if possible, increased their weight to *highest weight * similarity weight*. Consider a similarity weight $\frac{1}{3}$, all related classes ("Stroll") get increased their weight by 1. Furthermore all sub-classes of the highest weight classes get, if possible, increased their weight to *highest weight * similarity weight*. But in this example, "Stroll" has no sub-classes.

Now all classes that are relevant from the system's view have a weight (Shopping: 3, Stroll: 1, Jogging: 1). In a last step, each class weight gets divided by the sum of all class weights ($3 + 1 + 1 = 5$). So the final weights for the classes are: Shopping: 0.6, Stroll: 0.2 and Jogging: 0.2.

### 3.4.2 Extending the ontology

Algorithm 3 gets invoked by the presentation layer when a user wants to publish a new activity. It returns class suggestions for a specific activity Description. The algorithm

considers the strictness factor, see definition 16. Thus, not all classes that have a weight get returned. Only classes with a high weight get returned.

**Algorithm 3** getBestMatchingClasses(input_a: natural language activity description)

```
1: input_s ← strictness factor
2:
3: classes ← Set of classes (empty at the beginning)
4: matchingClasses = getClassesAndWeights(input_a)
5: classesCompleted = false
6: While ∑_{class∈classes} class.w ≤ input_s Do
7:    highestWeight = max_{∀classes∈matchingClasses} (class.w)
8:    ForEach class | class ∈ matchingClasses && class.w = highestWeight Do
9:       classes.add(class)
10:      matchingClasses.delete(class)
11:   EndFor
12: EndWhile
13: Return classes
```

Algorithm 4 gets invoked by the presentation layer in order to subordinate the individual to a specific class. The variable *flag* predicates whether the individual must be further subordinated (true) or if the individual was already automatically further subordinated (false).

**Algorithm 4** setIndividual(input_individual: individual to add, input_c class of individual, input_flag)

```
1: input_individual.flag = input_flag
2: input_c.addIndividual(input_individual)
```

If a user, let us call him "User U1", enters an language activity description in order to publish an activity and the system cannot return satisfying results, the user can subordinate this description (and thus an individual) to one top level class, for example to the class "Other Assistances". Then the flag will be set to "true". Suppose another user U2 who enters, in order to search for an activity, a description that matches the class "Zoo". This is the case if U2 chooses this class manually or if algorithm 3 just return this class. Suppose that U2 contacts User U1 by using an internal message service. Now algorithm gets invoked. Because U2's individual is subordinated to a subclass ("Zoo") of U1's "Other Assistances", the entered individual of User U1 will be moved to the class "Zoo" and the flag will be set to "false". A formal description of these procedure is described in algorithm 5.

**Algorithm 5** classifyIndividual(input_individual: individual to classify, input_i: all individuals in ontology, input_c classified class)

```
1: If input_individual.flag = true && input_c ⊑ input_individual.class Then
2:     input_i.delete(input_individual)
3:     setInvididual(input_individual, input_c, false)
4: EndIf
```

Algorithm 6 gets invoked by the application layer. The algorithm tries to find out whether a result quality is good or bad considering the quality factor. See also definition 17. Inputs for bad results can be sent to the software developer so that the algorithm or the ontology can be optimized. Moreover, if the result quality is bad, this algorithm can be used to suggest the user to use the faceted search.

**Algorithm 6** getResultQuality(input_a: natural language activity description)

```
1: input_qualityFactor ← quality factor
2: isGoodMatch ← boolean
3:
4: topLevelClasses = getTopLevelClasses
5: getClassesAndWeights(input_a)
6: ForEach tlc ∈ topLevelClasses Do
7:     tlc.w = ∑_{class⊑tlc} class.w
8: EndFor
9: If ∃ class ∈ topLevelClasses | class.w ≥ input_qualityFactor Then
10:    isGoodMatch = true
11: Else
12:    isGoodMatch = false
13: EndIf
14: Return isGoodMatch
```

# 4 Realization and Evaluation

## 4.1 Used Tools and Libraries

The desired part of the matchmaker was implemented by using Java 1.7. Furthermore some libraries were used. These are the OWL-API 3.4[11], which supports creating, manipulating and serializing owl ontologies. Furthermore the Pellet OWL2 reasoner[12] library was used. It makes it easier to get out information from the OWL2 ontology.

[11] http://owlapi.sourceforge.net/
[12] http://clarkparsia.com/pellet/

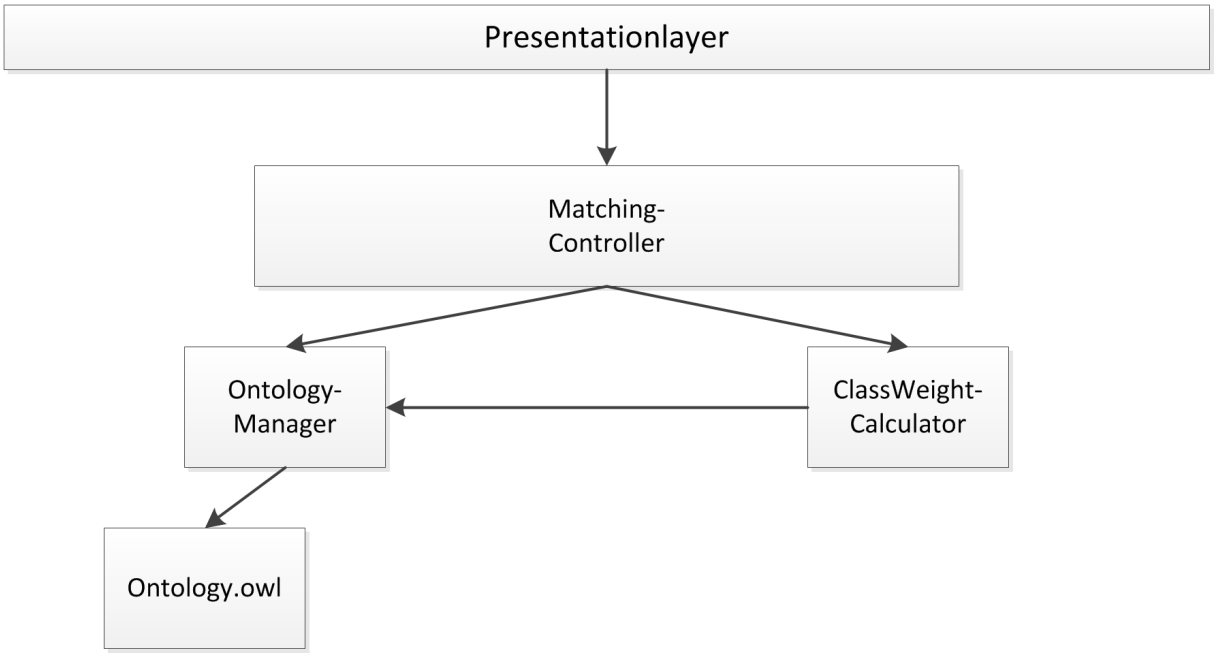



Figure 7: Basic structure of the implemented part of the matchmaker

For developing the java code Eclipse Indigo[13] was used and for creating the ontology Protégé[14] was applied. Figure 7 shows the three most important classes in the implemented matchmaker.
An arrow from class x to y means, that x holds a reference to an instance of y. All requests from the presentation layer will be sent to the MatchingController. The MatchingController will decide which method will be used. The OntologyManager is invoked for all requests that need information about the ontology. The ClassWeightCalculator is needed to calculate the weights. For calculating the weight, of course, it needs some information from the OntologyManager. Both, the ClassWeightCalculator and the OntologyManager are using the singleton pattern to make sure that only one instance of these classes can exist. A detailed description of the classes, for example information about class variables and methods, can be seen in the Javadoc which is in the project folder on the CD.

## 4.2 Direct Match

### 4.2.1 Requirements

In order to evaluate different variations of defining a direct match, there are defined some requirements for the direct match.

[13]http://www.eclipse.org/
[14]http://protege.stanford.edu/

**Distinction between active and passive activities** The direct matching algorithm should distinguish between doing an activity active (for example playing football) and doing an activity passive (for example watching football). The classes have not much background knowledge about an activity, so the direct matching algorithm will not be able to draw this distinction always. If the algorithm can not be sure if the description relates to an active or a passive activity it should not make a direct match to any class. Then the individual based algorithm, that have much more knowledge, should draw this decision.

**Considering the probability of the descriptions** The algorithm should not focus on the natural language activity descriptions that are unlikely to be entered. It should focus on the descriptions that are entered often.

**High precision** The precision is the probability that the natural language activity description relates to the returned class.
In order to get good results it is important in case of a direct match, that the returned classes are very likely to match the natural language description. If the algorithm has a low precision and thus makes many wrong matches, the user will not be satisfied with the results. So it is very important that class matches are very likely to be correct. The recall is the probability that a class that matches the natural language activity description is returned.
The recall is, of course, also important but it is less important than the precision. Because if the direct match algorithm will not return a class, then the individual based algorithm still will draw a decision.

### 4.2.2 Possibilities

There are many different possibilities how the input, a natural language activity description, can be compared to the classes in the ontology. The tables 2 - 6 show some different possibilities. Text in bold means that there was a match but there should not be a match or vice versa.

**Possibility 1: Description is substring of class in ontology**

| NLAD | Ontology | Match |
|---|---|---|
| football | watch football | match |
| ball[15] | watch football | **match** |
| watch football in the tv | watch football | **no match** |
| football watching | watch football | **no match** |
| play Football | watch football | no match |
| operavisit[16] | Opera | **no match** |

Table 2: Evaluation for the case that there is a direct match if the natural language activity (NLAD) is a substring of a name of a class in the ontology

| | Relevant | Not Relevant |
|---|---|---|
| **Match** | 1 | 1 |
| **No match** | 3 | 1 |

Precision: 0.5
Recall: 0.25

**Possibility 2: Class name is substring of the description**

| NLAD | Ontology | Match |
|---|---|---|
| football | watch football | **no match** |
| ball | watch football | no match |
| watch football in the tv | watch football | match |
| football watching | watch football | **no match** |
| play Football | watch football | no match |
| operavisit | Opera | match |

Table 3: Evaluation for the case that there is a direct match if a class name in the ontology is a substring of the natural language activity description (NLAD)

| | Relevant | Not Relevant |
|---|---|---|
| **Match** | 2 | 0 |
| **No match** | 2 | 2 |

[15] "ball" in the meaning of a dance event

[16] In the German language it is common to join two words. This is not common in the English language. Because the software is primarily made for the German-speaking users this problem is described by using the incorrect word "operavisit"

Precision: 1
Recall: 0.5

**Possibility 3: Class name is substring of the description or vice versa**

| NLAD | Ontology | Match |
|---|---|---|
| football | watch football | match |
| ball | watch football | **match** |
| watch football in the tv | watch football | match |
| football watching | watch football | **no match** |
| play Football | watch football | no match |
| operavisit | Opera | match |

Table 4: Evaluation for the case that there is a direct match if a name of the class in the ontology is a substring of the natural language activity description (NLAD) or vice versa

| | Relevant | Not Relevant |
|---|---|---|
| **Match** | 3 | 1 |
| **No match** | 1 | 1 |

Precision: 0.75
Recall: 0.75

**Possibility 4: One word of the description is also a word in a class name**

| NLAD | Ontology | Match |
|---|---|---|
| football | watch football | match |
| ball | watch football | no match |
| watch football in the tv | watch football | match |
| football watching | watch football | match |
| play Football | watch football | **match** |
| operavisit | Opera | **no match** |

Table 5: Evaluation for the case that there is a direct match if one word of the natural language activity description (NLAD) is exact the same word like one word of a class name

| | **Relevant** | **Not Relevant** |
|---|---|---|
| **Match** | 3 | 1 |
| **No match** | 1 | 1 |

Precision: 0.75
Recall: 0.75

#### Possibility 5: One word in a class name is a substring of the description

| **NLAD** | **Ontology** | **Match** |
|---|---|---|
| football | watch football | match |
| ball | watch football | no match |
| watch football in the tv | watch football | match |
| football watching | watch football | match |
| play Football | watch football | **match** |
| operavisit | Opera | match |

Table 6: Evaluation for the case that there is a direct match if one word in the class name is a substring of the natural language activity description (NLAD)

| | **Relevant** | **Not Relevant** |
|---|---|---|
| **Match** | 4 | 1 |
| **No match** | 0 | 1 |

Precision: 0.8
Recall: 1.0

### 4.2.3 Evaluation

Considering the requirement that the algorithm should, if possible, distinguish between active and passive activities, the possibilities 5 and 4 are not feasible. Considering the probability requirement, that says that the algorithm should focus on description that are entered often, the possibility 1 is also not feasible because possibility 1 is not able to match quite common activities like “football watching” or “watch football in the tv”. Furthermore it has a low precision and a very low recall. The possibilities 2 and 3 do not differ very much. Both cannot match “football watching”, possibility 2 can also not match “football” and possibility 3 makes a wrong match when the natural language activity description is “ball”. But it can be assumed that it is very unlikely

that somebody just enters the word “ball”. But it is much more likely that somebody enters “football”. Considering the amount of cases where possibility 3 is better than possibility 2, possibility 3 is used, in spite of the higher precision of possibility 2.

## 4.3 Individual based Matching

### 4.3.1 Irrelevant Words in the Individuals

The stop word list cleans many irrelevant words out of the natural language activity description, but there are still many words in the cleaned description that interfere assigning it to a class. This could be for example names of cities, dates and times, weekdays, often occurring words like “help” and so on. Figure 8 shows an ontology with individuals that contain irrelevant words. Moreover the weights for the natural language activity description “Who goes with me to a supermarket in Bamberg tomorrow?” are shown. Thus the cleaned description could be “goes supermarket Bamberg tomorrow“. The similarity weight is $\frac{1}{3}$.

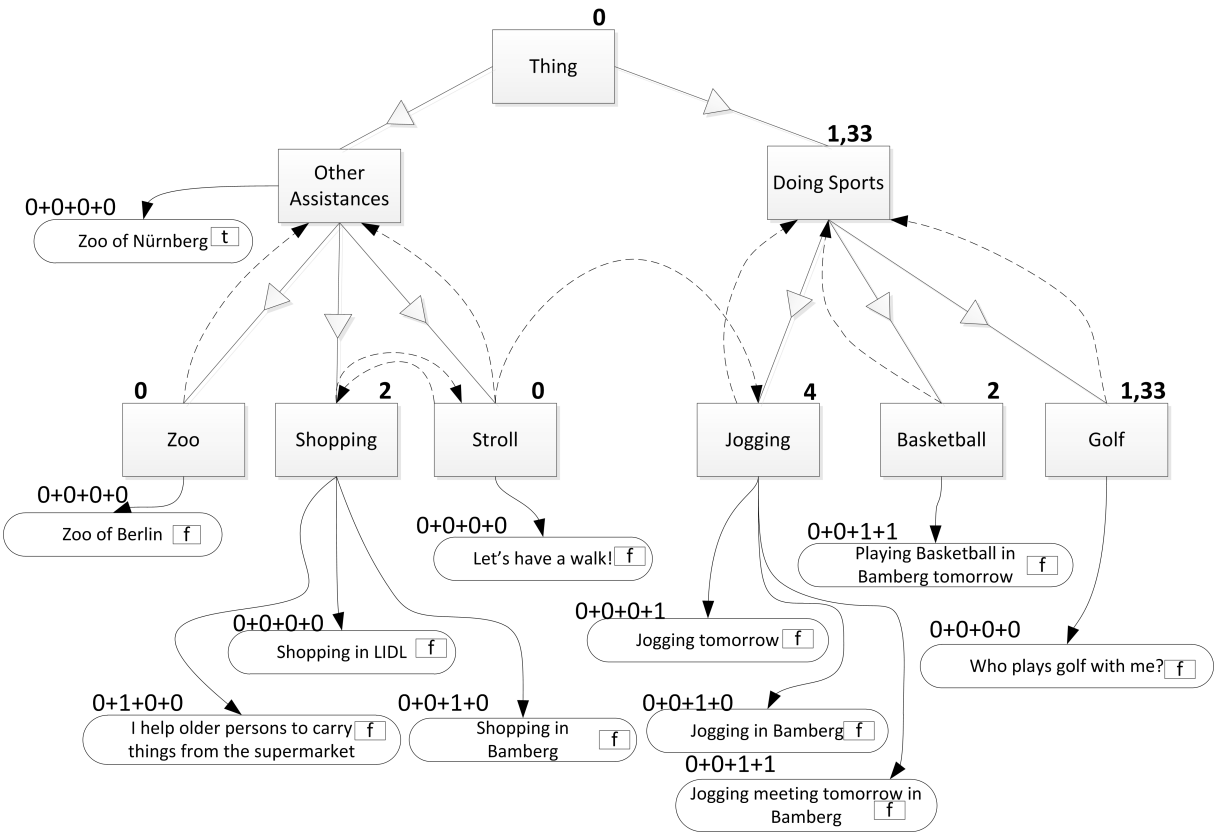



Figure 8: Ontology with weights for the description “Who goes with me to a supermarket in Bamberg tomorrow?”

The sum of all weights is 10.66. Thus we get following rounded normalized weights: Shopping: 0.19; Jogging: 0.38; Basketball: 0.19; Golf: 0.12; Doing Sports: 0.12.

#### Possible Solutions

Obviously, "Shopping" would be the correct class, but the algorithm returns many possible classes and the "Jogging" class has the highest weight. Following solutions could improve the results:

**Extending the stop word list** The stop word list can be extended by many words. This can be done manually or automatically. To automatically extend the stop word list, a new stop word can be defined by some restrictions, for example that it occurs in many individuals that belong to different classes. Alternatively these words can saved in an "unimportant word list". In contrast to the stop word list, these words are not ignored, but if these words match, then the weight of a class gets increased just very low.

**Using NLP** A natural language processing (NLP) algorithm can filter out words that have to do with a location or with time restrictions. This just alleviate the problem, because words like "help" and "game" or "play" still would mislead the algorithm.

**Motivate the User** The user could be motivated not to enter any time or location restrictions by developing a good interface. But this could lead to a low usability and would also, like the NLP, not solve all problems.

**Administrator takes care** One task of the system administrator could be to delete all irrelevant words from the individual. But the scope of this work would be to high, thus it is not feasible.

The best solution would be to extend the stop word list automatically. The algorithm can check, for example one time a day, if there are words in the individuals that belong to many different classes and add them to the stop word list. A disadvantage is that this algorithm just solve the problem afterwards (by adding words to the stop word list) after they were added to the ontology and not before. This means that it will take a some time until an algorithm recognizes that a word is not relevant for describing a certain activity. This solution can be improved by using a NLP algorithm. In some cases this can already extend the stop word list before a new individual gets added to the ontology.

### 4.3.2 Unbalanced Ontology

Figure 9 shows an unbalanced ontology, and the unnormalized weight of the classes when the description is "Watching the Brose Baskets game in a bar tonight", thus the cleaned one could be "Watching Brose Baskets game bar tonight". The similarity weight is $\frac{1}{3}$.

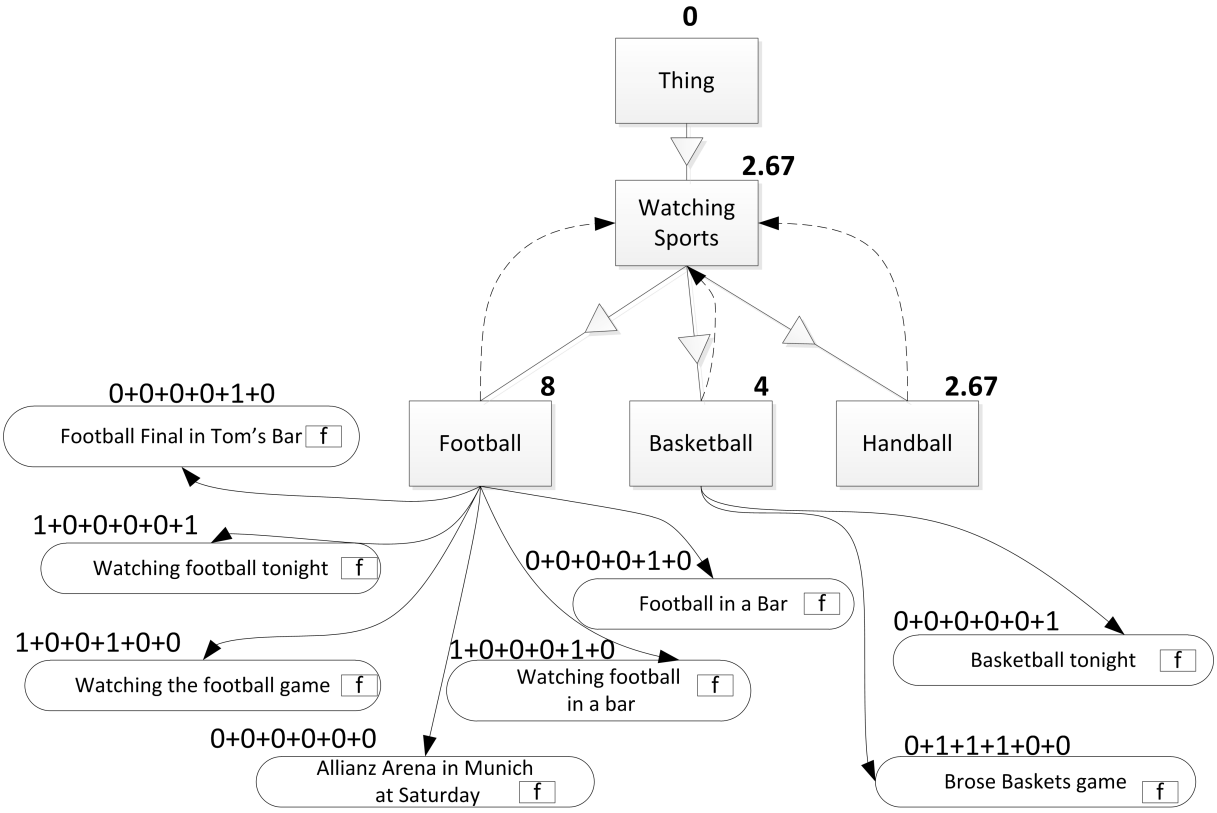



Figure 9: Ontology with weights for the description "Watching the Brose Baskets game in a bar tonight"

The sum of all weights is 17.34. Thus we get following rounded normalized weights: Football: 0.46; Basketball: 0.23; Handball: 0.15; Watching Sports: 0.15

**Possible Solutions**

"Basketball" would be the correct class, but the algorithm returns many possible classes and the "Football" class has the highest weight. This problem is very similar to the problem with irrelevant words in individuals. But the problem here is, that some words in the individuals, that cause the wrong match, are important. In this example, the word "watching" is, on the one hand important to distinguish if an activity is active or passive, but on the other hand it causes problems, because the class "Football" holds more individuals that contain "watching" than the class "Basketball" and thus, "Football' gets in some cases a higher weight.

**Eliminating active-passive distinction words** When the algorithm found out whether the natural language activity description relates to an active or a passive activity it simply can restart the matching algorithm without using the active-passive distinc-

tion word (for example "watching") and not considering all not matching classes (for example all active activity classes). It can be supposed that this algorithm could do a good work. But the implementation afford is quite high.

**Ignore whether an activity is active or passive** By ignoring if an activity is active or passive this problem will be the same like the problem of irrelevant words in the individuals and so could solved more easily. If it would be implemented in this way, the ontology can be restructured and the probability of a direct match becomes much higher. This would be a great advantage for the matching quality, but it won't be recognized any more if an activity should be done active or passive.

**Adjust the maximum weight of a class** Suppose the maximum weight of each class is one and suppose that if at least one word of an individual matches, the corresponding class increases its weight by $\frac{1}{Number\ of\ its\ individuals}$. In the example case in figure 9 the class "Basketball" would have the weight one ($2 * \frac{1}{2}$, because both individuals have at least one match) and "Football" would have the weight $\frac{5}{6}$, because all individuals except of "Allianz Arena in Munich at Saturday" have at least one match. Because of the has_Relation connection, "Basketball" and "Watching Sports" would have the weight 0.33. This approach would, on long-term, penalize classes / activities that can be described by using many different words. This is because such activities would have many individuals and it is very unlikely that a natural language activity description matches all of them. Thus this approach will cause more problems than it solves.

**Limit the number of individuals** There could be set a maximum of individuals each class can hold. So it can be restricted that a class can hold up to 15 individuals. On long-term, each class will hold 15 individuals. Thus a new individual could be saved only if another individual gets deleted. This would lead to a low background knowledge. Furthermore a quite complex algorithm must be developed that has to decide whether it is better to delete an individual in order to set a new individual or whether it is better not to save a new individual. But it can be assumed that an algorithm without much background knowledge cannot draw the decision which individual is better to keep in the ontology

**Each individual consists of exactly one word** Another approach is to save not more the whole natural language activity description in the individual but to save each word of an entered description in a separate individual. This means that a class holds many individuals and no duplicates. Entering the description "Watching football" and "Watching the football game" would create three individuals: "Watching" and "football" and "game". The main disadvantage of this approach is that words that are often used in a certain domain, for example "football" in the "Football" class, have the same weight like more unimportant words, for example "watching" or "game". Because of this disadvantage this approach should not be implemented.

To sum up, two of the described solutions could have success: The first mentioned solution describes to create another supporting algorithm: It finds out whether an activity description is active or passive and then runs the individual based algorithm and ignore the classes that are not relevant, because they are active / passive. Or the other solution that ignore active-passive distinction words by adding them to the stop word list and thus makes the problem of the unbalanced ontology to the same problem like described in chapter 4.3.1.

### 4.3.3 Wrong Assignment of an Individual

Algorithm 5 gets invoked when a user is satisfied with a specific offer after entering a natural language activity description. The user could be satisfied for example, when the user who searched get in contact with a user who published an activity. In some cases these approach could have a negative effect on the correctness of the ontology. These cases are:

**Intentional Manipulation** If a user knows the functionality of the matchmaker she or he can enter intentional an individual to a wrong top class and then further subordinate to an other class.

**Random Contact** A user X could contact another user Y after searching for an activity. But the published activity of user Y has nothing to do with the activity user X searched for. This could be for example if a user spontaneously gets interested in a published activity.

#### Possible Solutions

Obviously there must be made more complex restriction for algorithm 5. All automated solutions of this problem can be manipulated if somebody really wants it. There are some solutions that alleviate this problem.

- The system saves the IP address of the users and recognizes conspicuous activities of manipulating users. Disadvantage: Persons in an old people's home could have the same IP address. But it is not very likely that these persons are interested in manipulating the system. But still it is quite easy to change the IP address by using for example a proxy.
- There must be a certain number of different users that cause a further subordination of an individual - not just one user. Disadvantage: The published activity must be very popular and many persons have to contact the publisher of the activity to further subordinate an individual. This is not very likely.
- The system could be kept in this way and an administrator checks, for example one time a month, all subordination procedures, and deletes individuals if they

were wrong subordinated. Disadvantage: High expenditure of time.

All of these three solutions can be used simultaneously. This could alleviate the problem significantly. But it is likely that most of the individuals can be subordinated directly to a correct class and so this problem will not be severe. This is for example when a user enters “Trip to the Wagner opera” and the algorithm reason because of the word “opera” that it has to do with the class “Opera”.

## 4.4 Adjusting the static factors

There are three factors that can be changed: The similarity weight, the quality factor and the strictness factor. A high similarity weight would usually lead to a decreasing precision and to a high recall. However this cannot be generalized. Considering an activity description that was matched to a wrong class, a similarity weight could also increase precision and also increase recall.
A low quality factor will lead to the fact that the algorithm returns in most cases that the result quality is good and vice versa.
The strictness factor is used to return only the bast matching classes from the matchmaker’s view. So, if the algorithm returns that the matching quality is bad, it can suggest the best matching categories. According to how many categories should be suggested, the strictness factor must be set. A high strictness factor leads usually to a large number of returned suggestion and vice versa. However this also cannot be generalized, too. For example, if there are returned five classes by the weight calculating algorithm (algorithm 1), each class having a weight 0.2, always all classes will be suggested, except of the strictness factor is 0.0. See also algorithm 3
In this state there cannot be given a recommendation for setting these factors. All these factors must be adjusted in the test phase of the program. This is because these factors must be adapted according to how many persons are using the matchmaker and how good the matchmaker will work when getting natural language activities from elderly persons.

## 4.5 Ontology

The built ontology can be seen in the appendix and on the enclosed cd. For creating the ontology the choose of the class names is very important, because of the direct match. A class name must be, on the one hand, as short as possible to make it likely that a direct match is possible. On the other hand it must be as precise as possible. If a class name is called “Assistance” then all natural language activity descriptions that contain “assistance” will direct match to these class. This is usually not a good match, because there could be other, better matching classes. This is also very important in active-passive distinctions. Just calling an active activity “football” instead of “football playing” leads

to a match for example when somebody searches “football watching”.

Another possible improvement is to store other words as labels in a class to increase the probability of a direct match. OWL2 supports labels. Thus, a class could have the name “watching football” and this class have the labels like ”football stadium”, “Champion’s League” and so on. The algorithm can easily improved to consider these labels. Considering these example, a natural language activity description that contains “football stadium” will lead to a direct match in the class “watching football”. This will definitely improve the matching quality. This could be done automatically. An algorithm that recognizes that many inputs that belong to a certain class contain a substring that is not contained in individuals of other classes could add these substring as label automatically to the corresponding class.

Furthermore it is important to have just a small number of top-level classes. Persons sometimes should subordinate their desired activity to a top-level class. To make this procedure not too complicated there should be less and meaningful described top-level classes.

However, it could be better in the beginning phase, when less persons are using the matchmaker, to have an ontology that is not so deep and detailed. By having an ontology that is not so deep it is possible to keep a better overview of the ontology and the developer can examine if the matchmaker works well or not. In order to improve the result it is conceivable to use a string matching algorithm that compares the natural language activity to the activities entered in the database in order to change the weight of the published activities again.

# 5 Summary

The next step in the development of the matchmaker will be to implement the user interface, to implement the other parts of the matchmaker and to extend the ontology. When developing the other parts of the matchmaker it should be considered that the category of the activity is not always important. For example if somebody wants to go to a city that is far away, it could be less important what the persons want to do there, because the persons are already happy if they find somebody who is also going to this city.

The aim of this work is to present and implement an approach for a self-learning system. This approach will be part of a matchmaker. The matchmaker will match elderly persons with persons who have the same interests and elderly persons with volunteers. The part of the matchmaker that is developed in this work shall assign a natural language activ-

ity description to classes in an ontology. These classes represent activities. Furthermore the approach calculates a probability from the system's view that the natural language activity description belongs to a certain class. In addition, the approach makes use of similarities between activities. This means that these approach is also able to return activities that are similar to the desired activities.

The approach is completely implemented in java and can be tested. But there are still some problems, mentioned in chapter 4, that negatively impact the matching quality. If these problems will be solved or alleviated then the matchmaker will do a good work, also on long-term. Because of the better result quality it is, in general, better to try to increase the number of direct matches without decreasing the precision. This can be done for example by saving synonyms of the class names in the ontology and finding a better method of active-passive distinction. The problem of active-passive distinction makes direct matches in sport activities more unlikely.

# References

Banerjee, N., Chakraborty, D., Dasgupta, K., Mittal, S., Nagar, S., et al. (2009). Ruin?-exploiting rich presence and converged communications for next-generation activity-oriented social networking. In *Mobile data management: Systems, services and middleware, 2009. mdm'09. tenth international conference on* (pp. 222–231).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. In *Scientific american 284* (pp. 34–43).

Broder, A. (2006). *Workshop on facteted search.* access on 10th July 2012 from `https://sites.google.com/site/facetedsearch`.

Daconta, M., Obrst, L., & Smith, K. (2003). The semantic web: a guide to the future of xml, web services, and knowledge management.

Genesereth, M., & Nilsson, N. (1987). *Logical foundations of artificial intelligence* (Vol. 9). Morgan Kaufmann Los Altos, CA.

Li, L., & Horrocks, I. (2004). A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce*, *8*(4), 39–60.

Munz, M., Stein, K., Sticht, M., & Schmid, U. (2012). Matchmaking: How similar is what i want to what i get.

Raman, R. (2001). *Matchmaking frameworks for distributed resource management.* Doctoral dissertation, University of Wisconsin - Madison.

Söllner, M. (2011). *Integration von kategorialer und räumlicher information für einen quartiersbezogenen match-making service.* Bachelor's thesis, University of Bamberg.

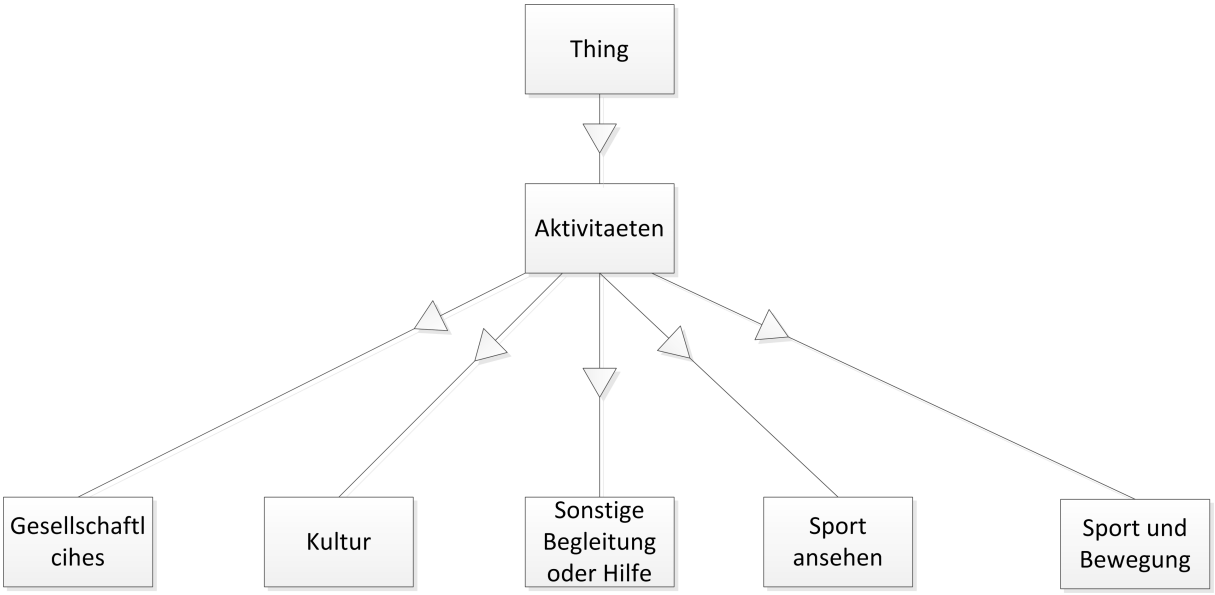
Staab, S. (2009). *Handbook on ontologies.* Springer Verlag.

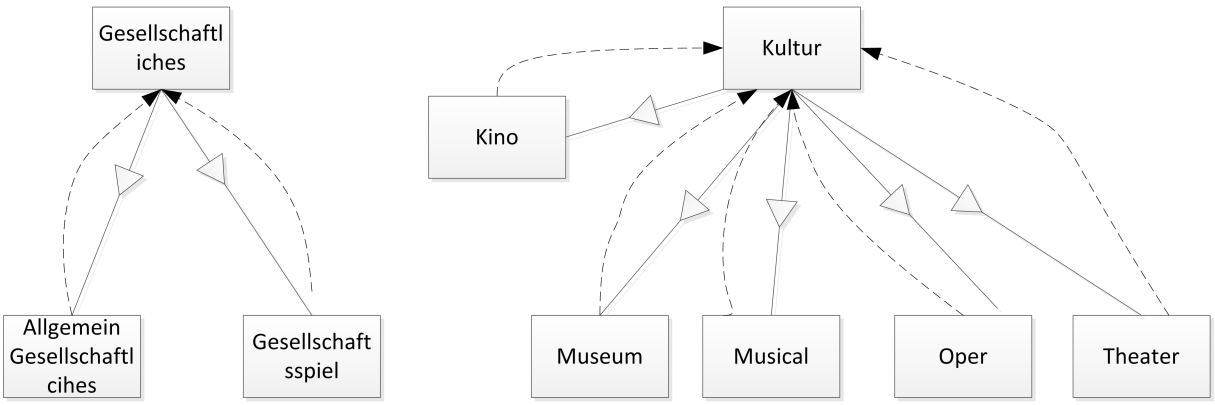
Stuckenschmidt, H. (2011). *Ontologien: Konzepte, technologien und anwendungen (informatik im fokus) (german edition).* Springer Verlag.

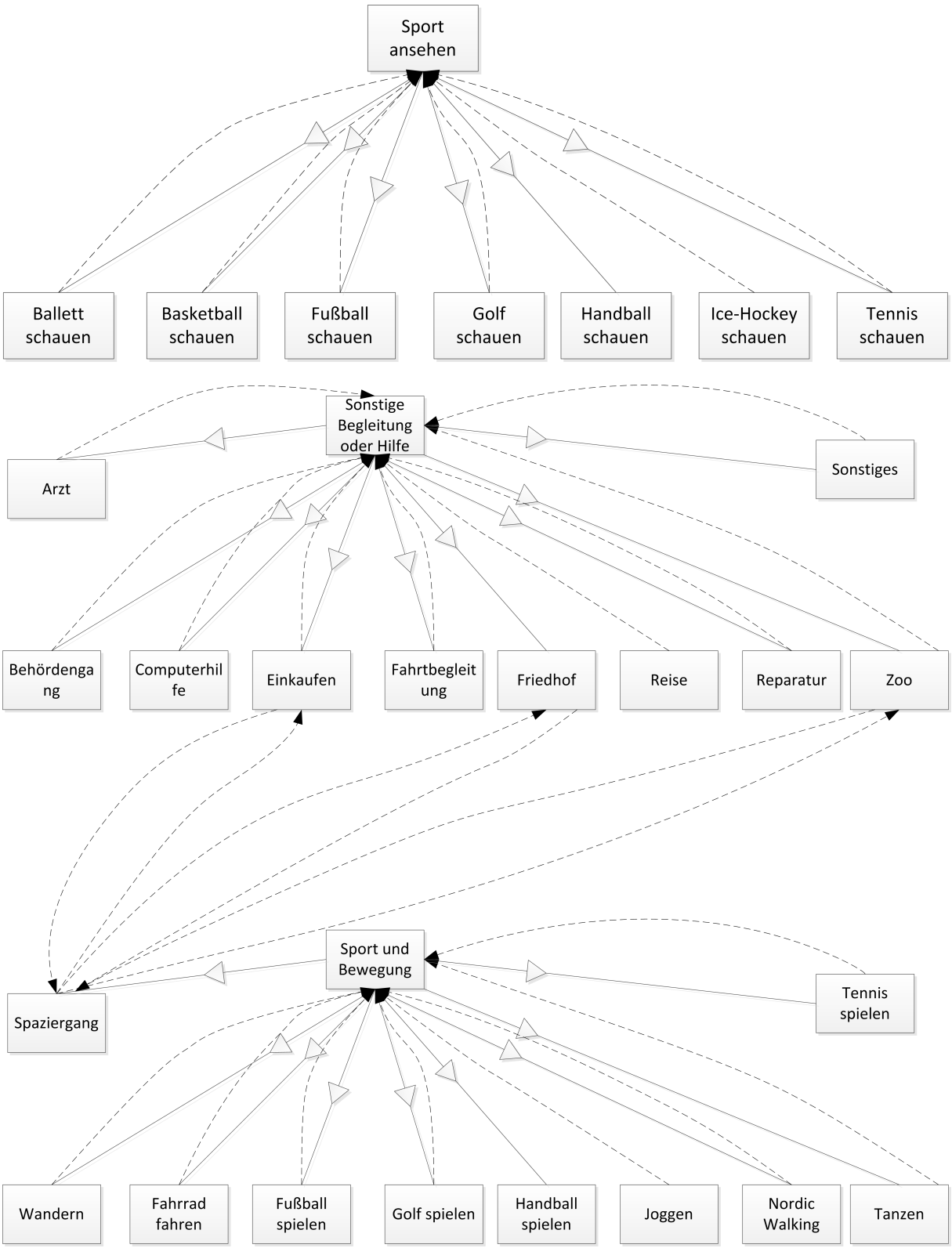
# A Appendix

## A.1 Ontology

Because the application of this matchmaker will be in Germany, the ontology is German. Note that this is just a draft of the ontology. This ontology does not hold any individuals. The ontology on the cd contains also some individuals.




Sport ansehen
Ballett schauen
Basketball schauen
Fußball schauen
Golf schauen
Handball schauen
Ice-Hockey schauen
Tennis schauen
Sonstige Begleitung oder Hilfe
Arzt
Sonstiges
Behördengang
Computerhilfe
Einkaufen
Fahrtbegleitung
Friedhof
Reise
Reparatur
Zoo
Sport und Bewegung
Spaziergang
Tennis spielen
Wandern
Fahrrad fahren
Fußball spielen
Golf spielen
Handball spielen
Joggen
Nordic Walking
Tanzen

## A.2 Implementation

This work comes with a CD. This CD contains the implemented part of the matchmaker including

- the source code,
- the ontology,
- the javadoc documentation, and
- the used libraries.

# Eidesstattliche Erklärung

Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

*Bamberg, 15.09.2012* *Lukas Berle*