

How similar is what I get to what I want – Matchmaking for Mobility Support

Ute Schmid, Lukas Berle, Michael Munz, Klaus Stein, and Martin Sticht

Faculty Information Systems and Applied Computer Science, University of Bamberg,
Germany

`{firstname.surname}@uni-bamberg.de`

Abstract. We introduce matchmaking as a specific setting for similarity assessment. While in many domains similarity is assessed between pairs of entities with equal status, in matchmaking there exists a source request which triggers search for the most similar set of available entities. We describe a specific scenario where elderly people request support or companionship for outdoor activities in the neighbourhood. The scenario is used to formulate requirements for a matchmaking framework. Similarity matching for support requests is based on hard criteria such as gender and on spatio-temporal constraints. Matching companions for outdoor activities needs a more sophisticated method taking into account semantic similarity of requested and offered activities.

1 Introduction

Cognitive scientists consider similarity to play a crucial role in most cognitive processes such as concept acquisition, categorisation, reasoning, decision making, and problem solving [1,2]. Major approaches to similarity in cognitive science as well as in artificial intelligence can be characterised on two dimensions: First, whether basic information about objects is metrical or categorial and second, whether objects are characterised by feature vectors or structural information [3,1]. The majority of cognitive and computational models of similarity are based on feature vector representations [1]. In contrast, many models of analogical reasoning are based on structure mapping since analogy making is per definition a transfer of relations from a base domain to a target domain [4,5].

In psychological research of similarity judgement, the typical task under investigation is that subjects are asked to rate similarity of two objects [1,6]. In this setting, the entities for which similarity is assessed play equivalent roles and can occur as first or second position during evaluation. Furthermore, entities are dissociated from the person who does the rating. However, there are many scenarios, where similarity between a “driver” entity and a series of candidates needs to be assessed. This type of similarity assessment is to the core of information retrieval research. Likewise, in analogical reasoning, a new target problem is the driver for retrieval of a suitable base problem. While analogical matching and transfer is based on structural similarity, for retrieval of a base problem many approaches propose feature based algorithms [7,8]. That is, for retrieval

problems, be it in analogy or information retrieval domains, the leading question is *how similar is what I get to what I need or want?*

In this paper we introduce matchmaking as a special domain of similarity-based retrieval. In general, matchmaking is the process of identifying similar or compatible entities. Requirements stated as a query by a user are matched with descriptions (e.g., of services or social events) provided by other users. We discern two different types of matchmaking – asymmetric and symmetric request relations. An *asymmetric relation* is constituted by a request for service. For example, somebody might look for a person who can drive her or him with a car to some destination. In this case, matching is based on features or constraints which are *complementary* for request and candidate entities. Complementary or fitting features are defined by a request/provides relation. A *symmetric relation* is constituted by a request for a person or entity with similar features or constraints. For example, somebody might look for a person who accompanies her or him to the cinema. In this case, matching is based on corresponding constraints, e.g. with respect to time and location, and similarity of interests.

In the following, we will first present some approaches to matchmaking which were proposed in different application domains. Afterwards, we introduce the domain for our matchmaking approach, that is assistance of outdoor mobility for senior citizens. The requirements are formulated and the matchmaking framework is described. Finally, we present our approach to activity matching based on self-extending ontologies.

2 Approaches to Matchmaking

There exists a wide range of application to matchmaking, such as (online) dating, sports, and business [9,10]. Approaches to matchmaking differ with respect to representation of data, reference to background knowledge and used similarity measures – resulting in different matchmaking algorithms.

In the following, we discuss three approaches to matchmaking. When formulating the requirements of a matchmaking system in the context of our application scenario, we refer back to these approaches – identifying concepts which can and which cannot be used for our specific setting.

2.1 Matching Resources With Semistructured Data

The classad matchmaking framework [11] is a centralised resource management system for managing distributed resources. It allocates information such as availability, capacity, constraints, and other attributes describing a resource. To describe a resource request or to announce a resource to the system, the information is represented as a so called classad (classified advertisement) – a *semi-structured data* model [12] comparable to records or frames in procedural programming languages. Classads are modelled via lists of pairs, each containing an attribute name and one or more values. Data pairs are used to describe offered and requested services. For example, when considering to use a workstation, a requester

would probably store information about the CPU’s capabilities or the disk space, while a provider offering a printing service would describe the printer’s throughput. In addition, constraints such as restricted user groups can be defined. Finally, ranking rules can be provided.

For a given request, the matchmaker tries to match the classad of the request to a resource with respect to any constraints given in the classads. The rule-based process evaluates expressions like *other.memory* \geq *self.memory*. The authors focus on the data structure and do not specify a specific matching algorithm. They state that the profiles can be matched in “a general manner” using the specified constraints. Additionally, as goodness criteria, the ranking rules can be applied to find out which classads fit better than others. Finally, the matchmaker informs the matched entities by sending the classads of each other. The resource provider decides whether it accepts or declines the request.

2.2 Matching Web Services Using Clustering

Fenza et al. [13] propose an agent-based system to match semantic web services. There are two different kinds of agents in the system: a broker agent (*mediator*) and one or more advertiser agents. A request for a service is handled only by the broker. When it encounters a request it converts it into a *fuzzy multiset* [14] representation. With these multisets a *relevance degree* is assigned to each possible ontology term that describes a web service according to the place, where the term occurs. For example, if the term occurs in the input specification of the service then it will get a relevance degree of 1. If it occurs in the textual description, then it will get a degree of 0.3 and so on. In this way, it is possible to weight the term for different occurrences via categories.

Advertiser agents interact with web services and with a single broker agent. Each web service description¹ is converted into a fuzzy multiset representation. Note, that the broker does the same with the user’s request. So in the end, a broker has a fuzzy multiset of a request and advertiser agents have a fuzzy multiset for each registered service. The broker sends the fuzzy multiset of the request to the advertiser agents to find an appropriate web service. If a web service matches with a request then the matched service is returned by the broker, the corresponding fuzzy multiset is stored to a central cluster and its job is done. Otherwise, the broker tries to find an approximate service by using a knowledge base which is divided into two distinct sets of knowledge: *static knowledge* and *dynamic knowledge*.

There are several ontologies modelled to specific domains in the static part of the background knowledge. To calculate an approximate match, the broker modifies the original request by utilizing the domain ontologies. The dynamic part of the knowledge consists of a cluster of fuzzy multisets where the web service descriptions of the known providers are stored (encoded as fuzzy multisets). It compares the fuzzy multiset of the modified request with the fuzzy multiset of each cluster center and selects the services most similar to the request. That is,

¹ A specific ontology for describing web-services named OWL-S is used.

services with the minimal distance to the request are candidates for an approximation. The similarity is therefore measured using the distances in the fuzzy cluster.

2.3 Matching Activities Using Ontologies

R-U-In? [15] is a social network primarily based on activities and interests of users. A user looking for company for an activity (e.g. going to the cinema or to a jazz club etc.) queries the system with a short description, including time and place. The matchmaker returns contacts found by the users' social network profile, who have similar interests and are located in close proximity. The found contacts need not be known by the querying user yet. For example, the new person might be a social-network "friend" of a "friend" identified by some social network service. Users can post their interests and planned activities on the platform in real-time, i.e. planned activities are dynamic and can often change at the last minute. As a result of this, participants in an activity get updates about changes immediately.

An ontology is used to realise the matching process. There are reasoning mechanisms for ontologies based on description logic [16] and therefore for ontologies based on OWL [17]. Banerjee et al. used an OWL-based *context model* for their activity-oriented social network. Interests are provided by the user itself and are based on tags. Each interest can be tagged via the dimensions *location*, *category* and *time*. Similar interests can be identified by matching on all dimensions: the time (e.g. *evening*, *8pm*, ...), the category (*horror movie*, *skating*, *jazz*, ...) and the location (*Bamberg*, *jazz-club*).

Tags entered by the user for describing or querying an activity are considered as concepts of the ontology. The matchmaker queries the context model which in return gives a set of similar tags. Those tags are then matched with the tags specified in the user profile. Based on the search criteria of a user, activities might match exactly or just partially. The search result of any match is then ranked by its geographical distance to the current location of the requesting user. Suppose, a user stores the activity (Park, skating, 3pm) and a second user searches for (skating, afternoon). While the activity *skating* is an exact match, *afternoon* matches only partially with *3pm*. As afternoon subsumes 3pm it is still possible to match the activity.

In general, the ontology is used to store background knowledge by modelling concepts and relations. For the presented prototype, this is done manually. After a query, the matching process is performed in two steps. First, the context-model is used to get semantically similar tags which are then compared to the tags of the other user's activity descriptions. However, details on how the tags are compared and matched and how the results are ranked (beside of the geographical distance) are not discussed by the authors.

3 SCENARIO

Maintenance of mobility in old age is a crucial factor of quality of life because many activities of daily living require being mobile [18]. Indoor mobility is important for independent living in one's own home, outdoor mobility is a prerequisite for activities such as shopping, visiting friends, and participating in social events [19]. Impairments of mobility result from different causes: old age often entails bodily handicaps such as restricted ability to walk or reduced eye-sight. However, mobility can also be affected by monetary or mental constraints such as the fear of falling [20] or the fear to travel alone with public transport [21].

Our research goal is to improve outdoor mobility and to establish social connections of (older) people. To improve mobility plus covering social aspects we have to distinguish between two domains of interest the matchmaker has to deal with: *mobility assistance* and *outdoor activities*, see also figure 1. Matchmaking in those two domains is either accomplished by a *best fit* approach or by a *similarity* approach. The idea is to build a platform mainly based on collaborative help, but also includes service providers.

In the best fit approach the matchmaker needs to determine the best 'help response' to a 'help request'. In the similarity approach the matchmaker needs to find related or *similar* content or activities.

3.1 Mobility Assistance

Mobility assistance tries to improve mobility of older people. Often, older people are in the position of needing temporal help. One can think of going to the supermarket, going to the medical doctor, using public transportations, carrying groceries, going for a walk around the block etc.. To improve mobility of those people the system can be used to request assistance. Volunteers then, can provide assistance and support in their mobility. We call the relationship between seniors and volunteers in the domain of mobility assistance *asymmetric*, because seniors depend on someone else (volunteer) who give support and help. Without the help of a volunteer a senior wouldn't be able to be as mobile as she likes to be or in the worst case couldn't realise it at all.



Fig. 1. *left: asymmetric.* A helping situation is an asymmetric relationship. The older person depends on someone who is willing to help, because she can't do it on her own. *right: symmetric.* The relationship of activities is symmetric. Involved parties are at an equal level, no one depends on someone else to do the activity.

In general, volunteers should be certified over some agency to guarantee they are trustworthy and qualified to deal with the specific needs of older people. Screening and training of volunteers implies some costs, seniors typically would be paying some small fee to be registered at the agency.

Scenario 1 - mobility assistance: asymmetric relationship. Mrs. Peters is a 82 years old lady who needs attendance in taking the public bus lines. She thinks it's to complicated for her, because she has to know how to buy a ticket, where to change bus lines, and the bus stations to get off. Lisa Gustafsson has an appointment at the medical doctor on Wednesday at 12.15 pm. She has a zimmer frame and needs to use the public bus to go to the doctor. Aylia Özdan is 61 and a volunteer. She is willing to help someone else on Monday, Tuesday, and Thursday from 11 am to 1 pm. Mr. Weber is a 58 years old invalidity pensioner. He has committed himself to work as a volunteer and is usually available from Monday until Friday from 12 pm to 16 pm.

In the situations above there are some information explicitly given and some implicitly. There are two people who have themselves committed to work as volunteers and there are two older people who need assistance. Naturally, one would match the help request of Lisa Gustafsson to Mr. Weber who is a volunteer. The time slot of Lisa Gustafsson is fixed, because she has an appointment at the medical doctor, and Mr. Weber's time slot as a volunteer matches in time and day. But the problem here is implicitly given. Lisa Gustafsson is a woman who is going to the doctor and she might be embarrassed about getting assistance from a man. Of course, Lisa Gustafsson could explicitly say she doesn't want a man as an assistance. In that case, the best fit would be Aylia Özdan, although the time slot doesn't fit.

The request of Mrs. Peters is less uncomplicated. She is afraid of taking the public bus, but whether a man or a woman can give her assistance in riding the bus is here of no importance. Therefore, possible matches are both Aylia Özdan and Mr. Weber, because there is no more information given to constraint the matching.

3.2 Outdoor Activities

With outdoor activities not only the mobility of older people can be improved, but also social life. Here, the system can be used to do activities together with other people. Seniors can search for other seniors who have *similar* interests in activities. An important difference between outdoor activities and mobility assistance (described in previous section) is, there are not necessarily volunteers needed. Instead, the focus here lies in meeting people about the same age and with similar interests. The relationship between seniors participating in an activity is *symmetric*, because no one depends on someone else to realize the activity. In the worst case, an activity could be done alone.

Scenario 2 - outdoor activity: symmetric relationship. Mr. Beck is 70 years old and interested in walking. He has also done Nordic walking in the past. None

of his acquaintances is interested in it, and he doesn't know any other person who has the same interests. Mrs. Novak is 73 years old and she likes all kinds of outdoor activities and to be outside and enjoy the nature. Mr. Miller, 81 years old, lives near a park. He loves to go for a walk in the park. He has a walking stick.

In this scenario there are three older people who are looking for other activity partners. When Mr. Beck searches for someone else who also likes walking, he finds that Mr. Miller likes it as well, because both activities are similar. But the system could also offer the activity of Mrs. Novak in the resulting list, because she likes to be in the nature and doing all kinds of activities. She hasn't provided any more information to the system, so the matchmaker can't constrain it any further. But all three activities are - at a higher level - similar and therefore match.

4 Requirements

From the scenarios described above various requirements arise which have to be considered in a matchmaking framework.

Activities A matchmaking framework has to deal with different kinds of requests when it comes to a matchmaking. The essence of scenario 1 is that someone is searching for help and someone else is offering a helping hand. Here, a request for help and an offer to help should be matched. In scenario 2 the situation is different. Users search for other users with similar interests. A matching service should match users with same or similar interests. All requests have in common that they concern "activities". As a result, requests are essentially a search for activities. Therefore, one requirement to a matching framework is to handle those activities properly.

While R-U-In? (Sec. 2.3) matches users with similar interests, it does not provide a fitting algorithm to match offers and requests of users. Classads (Sec. 2.1), on the other hand, supports different roles. It provides the data structure to model 'requests' and 'offers' but no matching algorithm. The fuzzy multiset approach (Sec. 2.2) supports the different roles of requester and provider as long as all parameters can be expressed in fuzzy multisets.

Constraints Activities are essentially sets of constraints. We distinguish between hard ("must") and soft ("should") constraints. If only one hard constraint can't be satisfied the whole activity won't be satisfied at all, as it is the case in scenario 1 not wanting a man as an assistance. If one soft constraint can't be satisfied, the activity will still be available as a possible match, as with Lisa Gustafsson and Aylia Özdan. In the context of matching similar activities, a soft constraint evaluated to false means matched activities do not fit so well. Note, what is seen as hard and soft constraints depends highly on user expectations. Scenario 2 (walking) is an example where a lot of soft constraints exist: time, place, day of week, and even the activity itself (walking, Nordic walking, doing an outdoor

activity). Whereas the examples of scenario 1 have a lot of hard constraints, like time, place, bus line, gender, and day of week.

The classad approach supports modelling of constraints, but the authors do not distinguish between hard and soft constraints. To overcome this, one could think of utilising annotations to distinguish categorically between hard and soft constraints. Both of the other approaches do not have explicit constraining mechanisms. The activity-oriented network R-U-In? models interests of users via three dimensions: “time”, “category”, and “location”. The model could be extended by an additional dimension specifying constraints. The downside of this approach is all dimensions of the model are represented by an ontology. That means, only the concepts of hard and soft constraints could be modelled, but no instances. Otherwise, constraints would be predefined and too inflexible. The fuzzy multiset approach directly supports (weighted) matching of soft constraints, but the data-structure has to be adapted to directly support hard constraints.

Roles The scenarios (Sec. 3) present two basic situations. On one hand, there are people needing help or searching for other people with same interests. On the other hand, there are people offering their help. But there are differences in the degree of helping someone. Some users do volunteering work and other users just do someone a favour. In the context of neighbourly help it’s important to distinguish between these, because there is a difference in the social commitment. the degree of social commitment can be modelled via roles. Roles can represent the different expectations users have, when searching for activities. For example, users searching for help expect to find someone offering help. The same is true vice versa. That is, a volunteer who is looking for users needing help expects to find posted activities.

As already stated, the goal of the proposed framework is to improve social life of older people by focusing on mutual assistance and neighbourly help. Therefore, the role of seniors represents those users who are searching for help or looking for company. The difference in helping is taken into account by other two roles. That is, volunteering work and doing favours are represented by separate roles, because they have different characteristics. People doing volunteering jobs do it on a regular basis and offer assistance explicitly. Usually, they have weaker conditions under which they are willing to help and try to be more flexible in scheduling an appointment with someone who needs help. Furthermore, they are willing to spend some of their spare time in helping others. Users doing favours don’t help out regularly and have stronger conditions under which they are willing to help someone. Doing someone a favour is usually a very time-limited act, so time is a hard constraint. Another characteristic of a favour is most people will do it only if it doesn’t interfere with their own plans and they don’t have to change their schedules.

Classads and fuzzy multisets are designed to match available resources to requests of resources. In those situations there exist only the two roles of service providers and service requesters, and no further differentiation is needed. An activity-oriented social platform like R-U-In? does only have one group of users. All users are interested in doing activities in their spare time. This leads to clear

expectations when using the platform. They either search for or post activities. Because of an activity-oriented user group only one role is needed to represent them.

Knowledge The matchmaking process can be improved by providing knowledge. It is suitable for matchmaking based on similarity to access background knowledge and user profiles.

Background knowledge is the general knowledge available in the system. Activities are represented by it and the knowledge is used to tell how similar different activities are. The matching service can offer similar activities by evaluating it (e. g. by taxonomic relationships). Suppose, someone searches for Nordic walking, but there is no direct match (as described in scenario 2). The matchmaking service can offer activities similar to walking instead and ignore other available activities (e. g. playing golf). Background knowledge has a disadvantage, though. It is often static and explicit. It doesn't change often and represents knowledge to a specific time. Moreover, updating static knowledge is often time consuming. To overcome this we consider to utilise user input as an additional dynamic source. It gives extra information to the matching process to examine the search query of a user and the chosen activity. Considering this, background knowledge is more dynamic and converges to requested user activities.

A user profile is also helpful in the matching process. It has two advantages: first, in the profile are those information stored a user normally doesn't want to re-enter every time a search is submitted. Second, information stored in the profile can be used to filter off matched activities which do not fit. In this way, the result list can be improved. Information in the profile could be among other things: interests of a users, trust to other users, constraints, and a user rating. Activities of other users should be withheld in the result list, if a user marked others as disliked or even untrusted. Trust and user ratings are really important in the context of neighbourly help and are valuable information in the matching process. A matching will get a much higher rating, if there already exists a relationship of trust between users. The implication is, they did some activities in the past, know each other and would like to do future activities together.

Classads [11] have in some extend a user profile, but they do not have any background knowledge. In classads only a resource can specify a list of trusted and untrusted requesters, so the relationship here is unidirectional. The activity network R-U-In? [15] uses both background knowledge and user profiles for a matching. While user profiles are updated in real-time, the background knowledge has no dynamic update mechanism so far. Moreover, there exists a policy repository where a user can define policies for participants when attending an activity. The downside of the platform is one can't rate users, can't mark them as liked or unliked, and it's not possible assigning any status of trust. In the fuzzy multiset approach [13] there is a distinction between background knowledge and fuzzy multisets. The background knowledge is realised in the form of domain ontologies and is static, according to the paper [13]. Whereas, fuzzy multisets are dynamic and are updated according to changes of services.

Requests The matching framework should be able to differentiate between two different classes of requests, *immediate request* and *stalled request*. They represent different searches of activities. Suppose, a user wants to do Nordic walking and issues a search. There aren't defined any preferences in the profile, like hard constraints. Further assume no exact match is possible, but there are two other activities stored (walking and enjoying nature), as the situation in scenario 2. As a result, the best matches are those two. The user has now the choice of either choosing any of the matches he or she is interested in by contacting the other person or to store the request in the system. A user should have the opportunity to store it, if he or she doesn't like any of the activities found or the results are not as expected

Every time a user initiates a new search for activities to the system he or she immediately receives all matching results best fitting the search. It is an immediate request. The result list is ordered according to a weighting so the best fitting activities are on top. In case the user isn't happy about the found matching results, he or she has the opportunity to initiate a stalled request. The request of the user is stored in the system's activity database and is from now on in monitoring mode. Depending on the preferences stored in the corresponding user profile the user will be notified about new activities of other users similar to his or her activity request. By utilising a stalled request one can find a match that best fits over a period of time. An immediate request returns matches that best fit to all currently available activities.

Classads [11] and the fuzzy multiset approach [13] match a request to the current available set of services only. They do not have to distinguish between different kinds of requests in their systems. In R-U-In? [15] users can search for and post activities. Activities are stored in a so-called activity group repository. A difference to our proposed matchmaking system is, stored activities are not in any monitoring mode, so users are not being informed about searches of other users. Rather, in R-U-In? a user will only be informed, if the requester is interested explicitly in an activity by sending him or her a message.

For the proposed system based on neighbourly help the described requirements are mandatory to the process of matchmaking. Because none of the approaches is appropriate for our needs we propose a matching framework with the required components.

5 COMPONENTS OF A MATCHMAKING FRAMEWORK

We introduce a framework with respect to the requirements identified in chapter 4. Figure 2 depicts all components of the proposed matchmaking framework. It shows the interaction between the components in which the matchmaker is the key component. A user searching for activities initiates a request to the system. All interaction between a user and the system happens via a mediator. The mediator decides whether it is an immediate request or a stalled request. If it's an immediate request the matchmaker will be called. For finding similar activities or

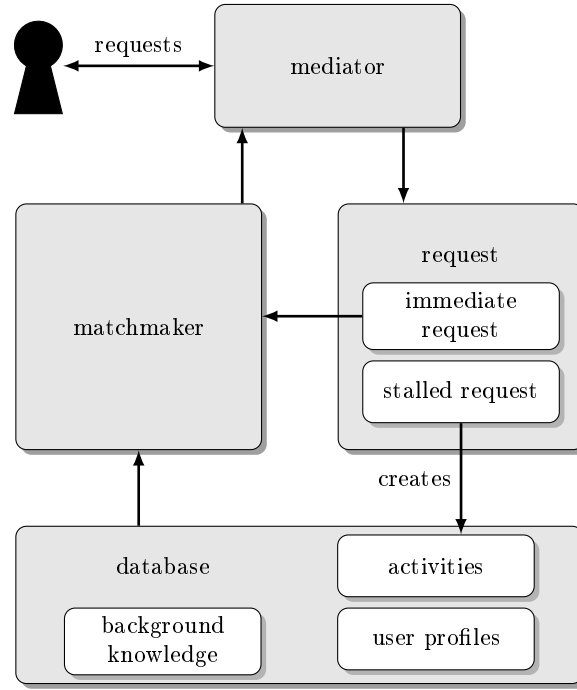


Fig. 2. Components of the matching framework and their interaction. The matchmaker is the key component of the system. It uses the underlying database for a matching and propagates the results to the mediator.

activities which fit to a given request the matching algorithm uses the underlying databases. That is, the background knowledge, the user profiles and the stored activities. A result list is then returned in response to the mediator. If the request is a stalled request an activity will be created in the activities database. There are two things to be aware of: first, the activity is in a monitoring mode. Second, a stalled request can only follow up on an immediate request. Whenever there is a new match for a stalled request the user will be informed.

5.1 Representing Constraints

Descriptions of activities as those mentioned in chapter 3 consist of features, such as gender, time, location, and the name of the activity itself. These features describing an activity are viewed as constraints for a matching and are classified by two dimensions:

similarity	\leftrightarrow	complement
hard constraints	\leftrightarrow	soft constraints

Some features need to be similar to the activity. Here, reflexivity of mapping holds. On the other hand, some features need to be complementary. For example,

the relationship between *needs car/offers car*. Here we speak of fitting and not of similarity. The matching can be modelled in such a way that the resulting scale corresponds to a similarity mapping. So both similarity and fitting can be processed together.

Hard constraints can be encoded using arbitrary complex boolean formulas on object properties while sets of *weighted* propositions are used for soft constraints. For example, let's assume that Mrs. Peters from scenario 1 only wants help from women who are at least 30 years old (hard constraint). This can be formalised as:

$$other.gender = female \quad \wedge \quad other.age \geq 30 \quad (1)$$

where *other* is a reference to a potential activity partner (similar to [11]). Consider the request of Mrs. Peters finding someone assisting her in riding the public bus as activity a_1 :

$$requires(a_1, assistance) \quad (2)$$

Requires relations are matched to corresponding *provides* relations of other activities. Say a_2 is given by another user, namely Aylia Özdan. Both relation will be used to check, if the activities fit, as:

$$provides(a_2, assistance) \quad (3)$$

The matchmaker must know that the relations *requires* and *provides* are matchable. However, matching two *requires* relations would not solve any problems. Whereas, relations of the same type (e.g. *likes*) would match in a similarity check:

$$likes(other, nordic\ walking) \simeq likes(other, walking) \quad (4)$$

Using constraints, time- and location-related restrictions can also be modelled. For time-related restrictions it's necessary to handle intervals to check temporal overlaps. Location-related restrictions calculate and weight distances. The distances are used for ranking purposes. Matches which have a shorter distance are better matches as similar pairs of matches but with greater distance.

5.2 Matching on Constraints

Checking hard constraints can be done by comparing the *requires* and *provides* relations of both, activities and the user profiles. If a hard constraint is violated by an activity description or an involved user profile, the activity will not be considered further in this query. Matching hard constraints should be done before soft constraints are considered. In this way hard constraints are used as filters to omit activities that are violated. Soft constraints have to be checked only on the remaining set of activities to calculate values of the matching quality.

Soft constraints have different weights, i. e. a value between 0.0 and 1.0 representing its importance to a user. These weights are either derived from the user profiles or from the user's query where the requester can specify the importance of each constraint.

If a soft constraint doesn't match, the matchmaker can

1. check the *severity* of the violation (e.g. the other’s age is 38 while the claimed age is 40; this violation would not be as strong as if the other’s age was 12). Note that this is only possible if a distance between the claimed and the actual value can be obtained (here difference in ages).
2. combine the severity of the violation with the weight of the constraint and find out how severe this violation is for the complete activity. Lower weights of constraints might qualify severe violations and vice versa.

If we assume that the severity can be normalised to a value between 0.0 and 1.0 where 0.0 means no violation and 1.0 represents the hardest possible violation, the *matching violation* V can be obtained by a sum

$$V = \sum_{c \in C} s_c \cdot w_c \quad (5)$$

where s_c represents the normalised severity of the violation of feature c and w_c the weight given by the user. C is the set of all relevant constraints.

In this way, it is possible to calculate for every remaining activity (after checking the hard constraints) a value of how well it fits to a query. A low V means a better fitting. According to these values, target activities can be ranked and presented in the corresponding order.

5.3 Knowledge from User Profiles and Missing Knowledge

We do not only distinguish hard and soft constraints, but also *profile constraints* and *on the fly constraints*. These constraints refer to where they are defined. Profile constraints are defined in user profiles and are used for recurring constraints only. If a user has defined constraints via the profile, the system will take them into account automatically when initiating a request. It’s a way of constraining the search implicitly. On the other hand, it should be possible to define constraints manually when doing a search. Those constraints are specified on the fly and are valid only for a specific request. Constraining the search manually should have higher priority as constraints specified in a user profile. For this reason, different knowledge has different priority. Information given in profiles has higher priority as background knowledge. A request has in turn higher priority as profiles. So knowledge with higher priority overwrites lower priority. As a result, constraints defined in the profile influence the search results implicitly, whereas constraints defined on the fly influence it explicitly.

Suppose, a user has the constraint *ignore(fitness walking)* in his profile and searches for a stroll in the city. Walk in the park, Nordic walking, and trekking pole are in the system as available activities. Because walking isn’t available, the only similar activities are a walk in the park, Nordic walking, and trekking pole. The matching service just offers a ‘walk in the park’ as an alternative activity and discards the Nordic walking and trekking pole, because they are on the ignore list via *ignore(fitness walking)*. Now suppose, the same user initiates an explicit request for Nordic walking. The request has higher priority as the

constraint in the profile and overwrites it. This approach allows users to find still activities explicitly, even when the profile states otherwise.

Further, it is also important not to treat unmatched constraints as fails because of missing information about the feasibility on the other side. Assume Lisa Gustafsson wants to go to an opera, but needs someone with a car to go there. So the car is a requirement that can be modelled as a (hard) constraint: *requires(car)*. Someone in her neighbourhood also wants to go to the opera. However, he doesn't mention in his stalled activity that he's going to drive with his own car. The problem here is, Lisa Gustafsson wouldn't find him although the activities would match. In this case, the matchmaker should identify the match and the missing fulfiller (car). Then, inform Lisa Gustafsson about the possible match and propose her to contact the person. After contacting him, Lisa Gustafsson is going to be able to go with him to the opera by car.

Missing information can be treated as *wild cards* which match everything. The matchmaker doesn't know if someone possesses a car, but the requirement is assumed to be fulfilled. However, the activity is going to be marked as *uncertain* until the person in question confirms he's going to the concert by driving his own car.

5.4 Presentation of Results

The approach we're going to use here is to present *all* matching results to the user. For this purpose, the result list is divided into three subsets: matches with complete information, matches with incomplete (missing) information, followed by matches violated by hard constraints. The results within the first subset are ranked by violation of soft constraints. Matches with no violations come first, then matches with low violation and finally matches with high violation. To improve the subset of matches with missing values the user is asked to provide additional information.

6 Activity Mapping

In the following we present details for symmetric matching of leisure activities. Since interaction with the matchmaking service should be as simple as possible, we decided against presenting selection menus for activities in – possibly hierarchical – categories and allow the user to input simple natural language descriptions instead.

Figure 3 shows a model that describes the process of activity suggestions. If a person searches for a partner to join some outdoor leisure activity, the database needs to be searched for a person who is interested in the same or a similar activity. As described in the section above there are further constraints to obey such as local nearness and similar time frames. For matching natural language descriptions we introduce a simple ontology which represents a hierarchy of leisure activities. That is, natural language descriptions are mapped either to a category

to which the current description is similar or to which the current description belongs with a certain probability.

The converter transforms the natural language activity description, the user profile and the restrictions to a Java object and sends this object to the category suggerter. The category suggerter figures out categories the description belongs to and sends them to the activity weight calculator. If the category suggerter cannot find out well matching categories it provides some category suggestions and the user returns his desired category. Finally, the activity weight calculator returns the best matching activities that are stored in the database to the user.

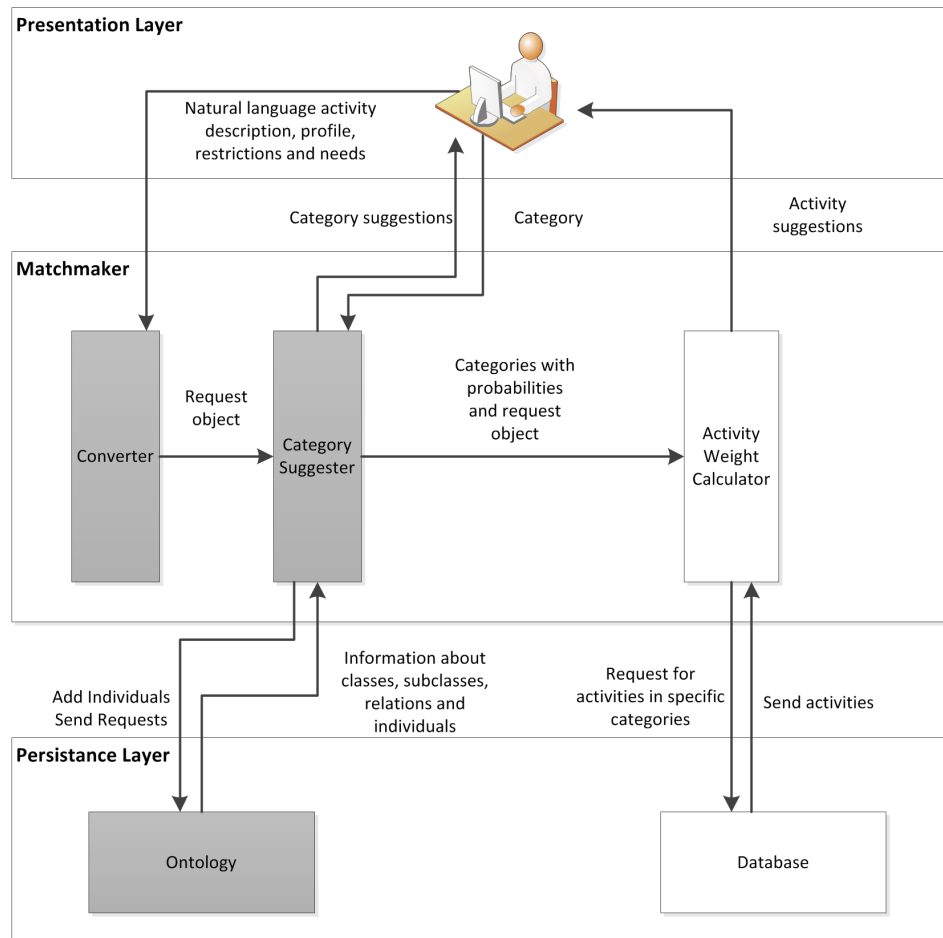


Fig. 3. The process of identifying activity suggestions (aspects discussed in this paper highlighted in grey)

The functional requirements with respect to the the category suggester and the ontology are:

Mapping Natural Language Activity Descriptions The application must be able to map natural language activity descriptions to activities. In addition it must be able to return a probability that a description fits to an activity and return how reliable the mapping is.

Semantic Matching The application must be able to deal with generalisations and similarities between different kinds of activities.

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

6.1 Ontology

One working definition for an ontology is “a formal, explicit specification of shared conceptualisation” [22]. A conceptualisation can be defined as “an abstract, simplified view of the world that we wish to represent for some purpose” [23]. Thus it is possible to model a part of the world by using an ontology. By contrast, a taxonomy is a hierarchical structure in which the objects are just related with one type of relation. This relation can be a sub-concept or a part-of relation [24]. In many ontologies there is a taxonomy which functions as backbone in order to create hierarchical structures [22]. The taxonomy is also a large and important part of the ontology for the desired matchmaker.

There are many different formal ontology languages that describe ontologies. We use OWL2, however making use only of a small part of the OWL2 features. These features are classes, individuals, self defined flags and self defined “has_Relation” annotations. In table 1 all used ontology elements of the matchmaker are described.

Table 1. Used elements in the matchmaker’s ontology





Representation	Definition	Example
	Direct subclass	“Other Activities” → “Zoo”
	Individual	“Zoo” → “Zoo of Berlin”
	has_Relation	“Stroll” → “Jogging”
	Flag	“Zoo of Berlin” → flag: false

Figure 4 depicts all elements which are used in the ontology for this matchmaker. The following definitions of ontology elements may differ from other definitions because the ontology elements are defined referring to their usage in this work.

Definition 1 (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 2 (Parent Class). *The parent class P of a class C is $P \mid [C \not\sqsubseteq P \wedge C \sqsubseteq P \wedge \neg \exists X (X \sqsubseteq P \wedge C \sqsubseteq X)]$.*

Definition 3 (Subclass). *The set of subclasses SC of the parent class C are all classes $S \mid (S \sqsubseteq C)$.*

Definition 4 (Direct Subclass). *The set of direct subclasses SC of the parent class C are all classes $S \mid (\text{parentClass}(S) \equiv C)$.*

Definition 5 (Top-Level Class). *A top-level class is a class that is a direct subclass of the class “owl:Thing” which is the root class in ontologies described in OWL.*

Definition 6 (Individual). *An individual is a natural language activity description that has been entered by a user and has been subordinated to a class. Individuals can hold a flag. When “a” is an individual that belongs to the class “C” then the formal description is $a : C$.*

Definition 7 (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 the flag is false, it already belongs to the correct class from the system’s view.*

Definition 8 (has_Relation). *Between two classes there can be a has_Relation connection. If activities in class B and their subclasses could be relevant for a user if class A is relevant, then there must be a “has_Relation” connection from class A to class B .*

6.2 The Algorithm

Definitions for the Algorithms

Definition 9 (Similarity Weight). *A similarity weight is a global weight between 0.0 and 1.0. After each class has received 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 10 (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.*

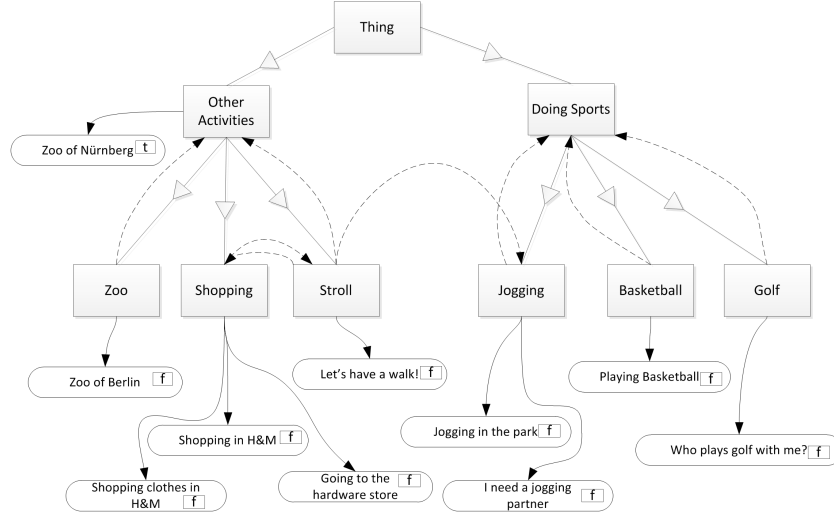


Fig. 4. An exemplary ontology

Definition 11 (Quality Factor). A quality factor is a global factor between 0.0 and 1.0. It is used to define whether a class suggestion for a specific natural language activity description is good or bad. Considering a quality factor x , a result is good if at least $100 * x$ % of the suggested classes are subordinated to one top level class, otherwise the result is bad.

Definition 12 (Direct Match). A direct match is a match that occurs if a natural language activity description is a substring of a class name or vice versa.

The technical definitions used in the algorithms are given in table 2.

Weight Calculating The algorithm calculates a normalised weight for the classes in the ontology. The normalised weights indicate the probability that a natural language activity description matches a specific class.

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

- 1: input_o \leftarrow set of all classes in the ontology
- 2: input_i \leftarrow set of all individuals in the ontology
- 3: input_k \leftarrow similarity weight
- 4:
- 5: directMatch = false
- 6: input_a = clean(input_a) //eliminates stop words
- 7: **ForEach** class \in input_o **Do**
- 8: **If** input_a \geq class.n **Then**

Table 2. Technical Definitions

C.n	If C is a class, C.n is its name
C.w	If C is a class, C.w is its weight
i.n	If i is an individual, i.n is its name
i.flag	If i is an individual, i.flag is its flag
i.class	If i is an individual, i.class is the class i is subordinated to. Formal: $i.class \equiv C \mid [i : C \wedge \neg \exists X \mid (X \sqsubseteq C \wedge i : X)]$
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 $\mid [C \not\sqsubseteq P \wedge C \sqsubseteq P \wedge \neg \exists X \mid (X \sqsubseteq P \wedge C \sqsubseteq X)]$
subClass(Class C)	Returns all S $\mid (\text{parentClass}(S) \equiv C)$
Class D \equiv Class F	$C \sqsubseteq D \wedge D \sqsubseteq C$
individualsOf(Class C)	Returns all i $\mid [i : C \wedge \neg \exists X \mid (X \sqsubseteq C \wedge i : X)]$
hasRelation(Class C)	Returns a set that contains all classes to which C is connected with a has_Relation connection
String s1 \geq String s2	Returns true if s1 is a substring of s2 or if s2 is substring of s1. Otherwise it returns false
a++	Adds 1 to the variable a. More formal: $a = a + 1$
Set.add(c)	Adds c to the set
Set.delete(i)	Deletes i from the set
C.addIndividual(i)	Adds individual i to the class C
getTopLevelClasses	Returns a set containing all C $\mid C \in \text{subClass}(\text{"Thing"})$

```

9:      /* There is a direct match. Set class weight to one and assign weights
      to subclasses and related classes */
10:     directMatch = true
11:     class.w = 1
12:      $\forall relClass \in hasRelation(c): \text{setWeightToClassAndSubClasses}(relClass,$ 
      input_k)
13:     If subClass(class)  $\neq \emptyset$  Then
14:         setWeightToClassAndSubClasses(class, 1)
15:     EndIf
16: EndIf
17: EndFor
18: If directMatch = false Then
19:     /* There is no direct match. Check for every word in the natural lan-
      guage description if it is a substring of a word in an individual. If it is
      a substring, increase the weight of the corresponding class */
20:     ForEach word  $\in$  getWords(input_a) Do
21:         ForEach class  $\in$  input_o Do
22:             ForEach individual  $\in$  individualsOf(class) Do
23:                 ForEach individual_word  $\in$  getWords(individual.n) Do
24:                     If word  $\geq$  individual_word Then
25:                         class.w++
26:                     EndIf
27:                 EndFor
28:             EndFor
29:         EndFor
30:     EndFor

```

```

31:   highestWeight =  $\max_{\forall \text{classes} \in \text{input\_o}}$  (class.w)
32:   ForEach class | class  $\in$  input_o && class.w = highestWeight Do
33:     /* Increase the weight of the subclasses and the related classes of the
       highest-weight-class(es) */
34:     setWeightToClassAndSubClasses(class, highestWeight * input_k)
35:      $\forall \text{relClasses} \in \text{hasRelation}(c)$ : setWeightToClassAndSubClasses(relClasses,
       highestWeight * input_k)
36:   EndFor
37: EndIf
38: /* Normalise the weights */
39: totalWeight =  $\sum_{\text{class} \in \text{input\_o}} \text{class.w}$ 
40:  $\forall \text{class} \in \text{input\_o}$ : class.w = class.w / totalWeight

end

```

Algorithm 2 *setWeightToClassAndSubClasses(input_c: class, input_w: weight)*

```

1: If input_c.w < input_w Then
2:   input_c.w = input_w
3: EndIf
4: ForEach class  $\in$  subClass(input_c) Do
5:   setWeightToClassAndSubClasses(class, input_w)
6: EndFor

end

```

The direct match (lines 7–17 in algorithm 1) is mostly relevant in the initial phase of the software usage because the direct match alleviates the problem of the cold start. The reason for this is that the ontology holds very few individuals in this phase and thus the algorithm has rather limited background knowledge. The algorithm must make use of the class names in the ontology.

If there is not a direct match for a given natural language activity description, the matchmaker will try to find out the related category by using the individuals (lines 18–30 in algorithm 1). Figure 4 illustrates an ontology with individuals.

Assumed that somebody enters the natural language activity description “I need to buy some modern clothes from H&M”. At first, all stop words of the description are deleted. So the cleaned description could be “Need buy modern clothes H&M”. Then the algorithm checks if a direct match is possible. Considering the direct match definition (definition 12), there is no direct match in the ontology in figure 4.

The next step is to start the individual based matching: It assigns weights to potential matching classes. If a word in the description is a substring of a word in an individual or vice versa, the class weight of the individual’s class is increased by one. This is done for each word in the description and for each word in the name of every individual. After finishing this process, all potentially

relevant classes have a weight. The result of this process can be seen in figure 5. 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 (here: “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 is the sum of the sums of its belonging individuals.

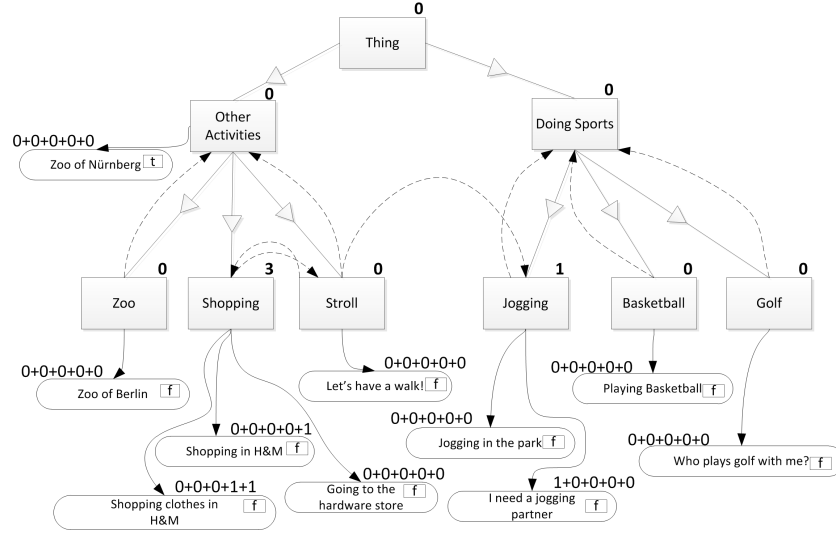


Fig. 5. Ontology with weights for the natural language activity description “I need to buy some modern clothes from H&M” (“has_Relation” connections are not considered)

In the next step, the class(es) that have the highest weight (here: “Shopping”) are in the focus. The weight of all classes and their subclasses that are connected with a “has_Relation” connection from the highest weight classes increase, if possible, their weight to *highest weight * similarity weight*. Considering a similarity weight of $\frac{1}{3}$, the weight of all related classes and their subclasses (here it is just “Stroll”) is increased by one. Besides the weight of all subclasses of the highest weight classes is increased to *highest weight * similarity weight*. In this example, “Shopping”, however, has no subclasses.

Now all classes that are relevant from the system’s view have a weight (Shopping: 3, Stroll: 1, Jogging: 1). In the last step the class weights are normalised. This is done by dividing each class weight 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.

Extending the ontology Algorithm 3 is 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 10. Thus only classes with a high weight are returned.

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

```

1: input_s  $\leftarrow$  strictness factor
2:
3: classes  $\leftarrow$  set of classes (empty at the beginning)
4: matchingClasses = getClassesAndWeights(input_a)
5: classesCompleted = false
6: While  $\sum_{class \in classes} class.w \leq input\_s$  Do
7:   highestWeight =  $\max_{class \in matchingClasses} (class.w)$ 
8:   ForEach class | class  $\in$  matchingClasses && class.w = highestWeight
     Do
9:     classes.add(class)
10:    matchingClasses.delete(class)
11:   EndFor
12: EndWhile
13: Return classes

```

end

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

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

```

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

```

end

If a user (“U1”) enters a natural language activity description in order to publish an activity and the system cannot return satisfying results, the user can subordinate this description (and in this way an individual) to one top level class, for example to the class “Other Assistances”. Then the flag will be set to “true”. Assumed another user (“U2”) enters, in order to search for an activity, a description that matches the class “Zoo”. It matches the class “Zoo” if U2 chooses this class manually or if algorithm 3 just returns this class. Supposed that U2 contacts U1 by using an internal message service, algorithm 5 is invoked. Because U2’s description belongs to a subclass (“Zoo”) of U1’s “Other Activities”,

the entered individual of U1 is moved to the class “Zoo” and the flag is set to “false”. A formal description of this procedure is described in algorithm 5.

Algorithm 5 *classifyIndividual(input_individual: individual to classify, input_c: classified class)*

```

1: input_i  $\leftarrow$  set of all individuals in ontology
2:
3: If input_individual.flag = true && input_c  $\sqsubseteq$  input_individual.class
   Then
4:   input_i.delete(input_individual)
5:   setIndividual(input_individual, input_c, false)
6: EndIf

end

```

Improving the ontology To increase the probability of a direct match, a possible improvement is to store other words as labels in a class. Accordingly a class could have the name “watching football” and this class has labels like ”football stadium”, “Champion’s League” and so on. The algorithm can easily be improved in order to handle labels. Considering the mentioned example, a natural language activity description that contains “football stadium” will lead to a direct match with the class “watching football”. This will definitely improve the matching quality. The creation of labels could be done automatically. An algorithm can be developed that recognises that numerous individuals of a certain class contain a specific substring which cannot be found in the individuals of other classes. Then the algorithm can add the substring automatically as label to the corresponding class. Figure 6 shows such an automated ontology transformation.

7 CONCLUSION AND FUTURE WORK

We proposed a framework for matchmaking for asymmetric support requests and for joint activity requests. This framework constitutes the core of a community-based platform for elderly people. It is part of a larger web-application developed in the interdisciplinary research project EMN-Moves. While asymmetric matching of support requests can be dealt with in a rather simple manner, activity matching relies on a more complex approach based on ontologies.

The proposed ontology-based matching algorithm could also be used as retrieval component in specific settings of analogy making to incorporate background knowledge not only for mapping [?,?] but also for finding a suitable source problem.

Currently, the first prototype of our matchmaking framework is tested in a social service company.

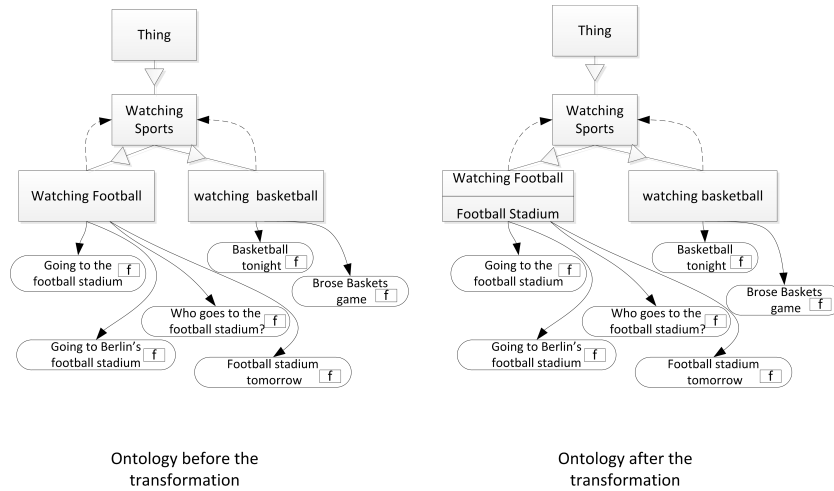


Fig. 6. The automated creation of labels

Acknowledgement. This work is funded by BMBF grant 16SV5700K (Technology and Innovation), Cooperation project “Europäische Metropolregion Nürnberg macht mobil durch technische und soziale Innovationen für die Menschen in der Region” (EMN-MOVES). We thank members of the senior citizen councils of Bamberg, Erlangen, and Nürnberg. We also thank the reviewers for their helpful comments.

References

1. Goldstone, R.L., Son, J.Y.: Similarity. *Psychological Review* **100** (2004) 254–278
2. Gentner, D., Markman, A.: Defining structural similarity. *The Journal of Cognitive Science* **6** (2006) 1–20
3. Schmid, U., Siebers, M., Folger, J., Schineller, S., Seuß, D., Raab, M., Carbon, C.C., Faerber, S.: A cognitive model for predicting esthetical judgements as similarity to dynamic prototypes. *Cognitive Systems Research* **24** (2013) 72–79
4. Gentner, D.: Why we’re so smart. In Gentner, D., Goldin-Meadow, S., eds.: *Language in Mind*. MIT Press (2003) 195–235
5. Schmid, U., Wirth, J., Polkehn, K.: A closer look on structural similarity in analogical transfer. *Cognitive Science Quarterly* **3**(1) (2003) 57–89
6. Tversky, A.: Features of similarity. *Psychological Review* **85** (1977) 327–352
7. Thagard, P., Holyoak, K., Nelson, G., Gochfeld, D.: Analogical retrieval by constraint satisfaction. *Artificial Intelligence* **46** (1990) 259–310
8. Forbus, K., Gentner, D., Law, K.: MAC/FAC: A model of similarity-based retrieval. *Cognitive Science* **19**(2) (1995) 141–205
9. Whitty, M.T., Baker, A.J., Inman, J.A., eds.: *Online matchmaking*. Palgrave Macmillan, Basingstoke (2007)

10. Shaw, D., Newson, P., O'Kelley, P., Fulton, W.: Social matching of game players online (2005)
11. Raman, R., Livny, M., Solomon, M.: Matchmaking: Distributed resource management for high throughput computing. In: High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on, IEEE (1998) 140–146
12. Abiteboul, S.: Querying Semi-Structured Data. In: Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings. (1997) 1–18
13. Fenza, G., Loia, V., Senatore, S.: A hybrid approach to semantic web services matchmaking. *International Journal of Approximate Reasoning* **48**(3) (2008) 808–828
14. Miyamoto, S.: Information clustering based on fuzzy multisets. *Inf. Process. Manage.* **39**(2) (March 2003) 195–213
15. Banerjee, N., Chakraborty, D., Dasgupta, K., Mittal, S., Nagar, S., et al.: R-U-In?-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, IEEE (2009) 222–231
16. González-Castillo, J., Trastour, D., Bartolini, C.: Description Logics for Matchmaking of Services. In: IN PROCEEDINGS OF THE KI-2001 WORKSHOP ON APPLICATIONS OF DESCRIPTION LOGICS. (2001)
17. Horrocks, I., Patel-Schneider, P.: Reducing OWL Entailment to Description Logic Satisfiability. In Fensel, D., Sycara, K., Mylopoulos, J., eds.: *The Semantic Web - ISWC 2003*. Volume 2870 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg (2003) 17–29
18. Gagliardi, C., Marcellini, F., Papa, R., Giuli, C., Mollenkopf, H.: Associations of personal and mobility resources with subjective well-being among older adults in Italy and Germany. *Archives of Gerontology and Geriatrics* **50** (2010) 42–47
19. Mollenkopf, H.: Enhancing Mobility in Later Life: Personal Coping, Environmental Resources and Technical Support, the Out-Of-Home Mobility of Older Adults in Urban and Rural Regions of Five European Countries. Ios Press (2005)
20. Scheffer, A.C., Schuurmans, M.J., van Dijk, N., van der Hooft, T., de Rooij, S.E.: Fear of falling: measurement strategy, prevalence, risk factors and consequences among older persons. *Age and Ageing* **37**(1) (2008) 19–24
21. Hausteil, S., Kemming, H.: Subjektive Sicherheit von Senioren im Straßenverkehr. *Zeitschrift für Verkehrssicherheit* **54**(3) (2008) 128–133
22. Staab, S.: Handbook on ontologies. Springer Verlag (2009)
23. Genesereth, M., Nilsson, N.: Logical foundations of artificial intelligence. Volume 9. Morgan Kaufmann Los Altos, CA (1987)
24. Daconta, M., Obrst, L., Smith, K.: The Semantic Web: a guide to the future of XML, Web services, and knowledge management. Wiley (2003)