Matchmaking: How similar is what I want to what I get

Michael Munz and **Klaus Stein** and **Martin Sticht** and **Ute Schmid** [1]

Abstract. We introduce matchmaking as a specific setting for similarity assessment. While in many domains similarity assessment is between pairs of entities with equal status, this is not true for matchmaking in general. Usually, in matchmaking there exists a source request which triggers search for the most similar set of available entities. Whether an entity is acceptable depends highly on the application domain. We describe a specific scenario where elderly people request support or companionship for activities away from home. The focus is primarly based on neighbourly help, like helping someone to carrying his or her shoppings, finding someone else to enjoy a performance or simply for taking a walk around the block. Then, the scenario is used to formulate requirements for a matchmaking framework and for the matchmaking service.

1 INTRODUCTION

Cognitive scientists consider similarity to play a crucial role in most cognitive processes such as concept acquistion, categorization, reasoning, decision making, and problem solving [5, 4]. Major approaches to similarity in cognitive science as well as in artificial intelligence can be characterized on two dimensions: First, whether basic information about objects is metrical or categorial and second, whether objects are characterized by feature vectors or structural information [10, 5]. In psychology, the typical task under investigation is that subjects are asked to rate similarity of two objects. In this setting, the entities for which similarity is assessed play equivalent roles and often 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 and can be characterized by the questions *how similar is what I want to what I get*?

In this paper we introduce matchmaking as a special domain of information 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. Typically, a good match is obtained by identifying features or constraints which are *similar* and – in addition – by features or constraints which are *complementary* for request and candidate entities. Complementary or fitting features are defined by a request/provides relation.

There exists a wide range of application to matchmaking, such as (online) dating, sports, eSports and business [12, 11]. In those domains, the matching process is based on different assumptions about what "similarity" means. In (online) dating a matchmaker tries to bring together people with similar interests or similar personality. Whereas in the area of sports a matchmaker has to consider the skills and competence of sportsmen and of teams when it comes to a matching. In business, a matchmaker could have the job of finding appropriate services for a request. Here, similarity depends on what kind of service one is interested in. The examples are all from different domains, that means finding something that is similar to a request depends on the domain of application.

The paper is organised as follows: first, we review three different approaches of matchmaking applied to different domains. Then we present two different kinds of scenarios, where older people search support or compagnions for activities. The scenarios are used to derive requirements for a matchmaking framework. In chapter 5 we present the components of framework and conclude with a short discussion and future plans. The focus of this paper is on the presentation of the system architecture (backend) by which matchmaking can be realized. We are not concerned with the user interface (frontend).

2 APPROACHES TO MATCHMAKING

In this section we discuss three existing approaches for matchmaking with respect to four major questions:

1. How is the data (advertisements, queries) represented?
2. Does the approach make use of background knowledge?
3. Which matching algorithm is applied?
4. Which fitting measurement is used?

The described approaches are applied in different domains. The approaches [9] and [3] are related to the business domain whereas [2] is related to dating and meeting people. Furthermore, they have a different understanding of what actually similarity means as previously discussed. It is this difference that drives the matchmaking process in different directions.

2.1 Matching Resources With Semistructured Data

The classad matchmaking framework [9] is a centralized resource management system for managing distributed resources. It allocates information, like availability, capacity, constraints, and other attributes describing a resource. Those information are used in the matchmaking process to find a proper match. The idea here is to use classads (classified advertisement), a *semi-structured data* model [1] comparable to records or frames, to describe a resource request or to announce a resource to the system. Classads are modelled via lists of pairs, each containing an attribute name and one or more values, to store semi-structured data. 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. It's possible to define

[1] University of Bamberg, Germany, WIAI, email: {name.surname}@uni-bamberg.de

constraints, restricted user groups and rules to rank each other. Both, service provider and service requester use classad descriptions. This makes it easy to compare the query with the suppliers' offers, looking at similarities of attributes and constraints and to rank the offers found in this process.

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 >= 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. Unfortunately, further details are not given by the authors.

Finally, the matched entities will be informed by sending them the classads of each other by the matchmaker and the resource provider decides to accept or decline the given request.

2.2 Matching Activities Using Ontologies

R-U-In? [2] 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 system with a short description, including time and place. The matchmaker returns contacts found by the user's 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 [6] and therefore for ontologies based on OWL [7]. 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*. In this way, one can find similar interests by matching on all dimensions: the time (e.g. *evening*, *8 pm*, ...), 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, 3 pm) and a second user searches for (skating, afternoon). While the activity *skating* is an exact match, *afternoon* matches only partially with *3 pm*. As afternoon subsumes 3 pm 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.

2.3 Matching Web Services Using Clustering

Fenza et al. [3] propose an agent-based system to match semantic web services. There are two different kinds of agents in the system: a broker agent (kind of *mediator*) and one or more advertiser agents. A request for a service is handled only by the broker itself. When it encounters a request it converts it into a *fuzzy multiset* [8] 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 description[2] 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 web 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 approximation, the broker modifies the original request by utilizing the domain ontologies. The dynamic part of the knowledge consists of the 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, 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.

3 SCENARIO

Many older people at a specific age often don't leave their home on their own, because of several factors: they might be more anxious in late life or may have physical health problems. They also might be more socially isolated, have significant changes in living arrangements, the loss of mobility, fewer flexibility, and loss of their independance. All this factors contribute to withdrawal from social life and thereby reduce quality of life.

To have an independent life at an old age mobility is crucial to being active and to stay in contact with other people. Therefore, the goal is to improve mobility and social connections of (older) people. That is, to bring together people who do need help, but also people who want to meet others and people who offer help.

The idea is to build a platform mainly based on collaborative help, but also includes service providers. In this paper we focus on the matchmaking framework of the platform. The context of collaborative help means to match people asking for help to people offering help and vice versa. People looking for help are going to be mostly elderly people and people offering help are going to be mainly volunteers.

[2] A specific ontology for describing web-services named OWL-S is used.

In the following, we are looking at two kinds of scenarios the system might be confronted with. They represent two different approaches the matching service has to deal with. The first scenario describes matchmaking based on best fit, while the second scenario describes matchmaking based on similarity. Fitting and Similarity are discussed in more detail in chapter 5.

Scenario 1 (a) *Mrs. Weber is looking for a babysitter for her 3 years old daughter on weekends on a regular basis from 1 pm until at least 4 pm.* (b) *Mrs. Peters is an 82 years old lady who needs attendance in taking the public bus lines. She thinks it's too complicated for her, because she has to know how to buy a ticket, where to change bus lines, and the bus station to get off.* (c) *Zoey plays guitar and goes to the music lessons every Wednesday after school at 3 pm and takes the bus line 901. She would agree in attending someone else.* (d) *Aylia Özdan is 31 and will help someone else on Sundays if it's between 11 am – 8 pm."*

The examples provide some information: Zoey works as a volunteer every now and then. She would accompany someone else under some conditions. She is using the bus line 901 at a very specific time (3 pm). So the person to accompany should use the same bus line and should be there before the bus arrives at the bus stop. If these conditions are met, Zoey will accompany a person until the bus arrives at the bus stop she has to get off. The request of Mrs. Peters will match this offer, if she also uses bus line 901 and waits at the same bus stop.

The request of Mrs. Weber looking for a babysitter on weekends matches only partial with the offer of Aylia Özdan to help someone on Sundays. Note, there is a difference between the help of Zoey and Aylia Özdan. While Zoey is helping the old lady by courtesy as long as it doesn't interfere with her plans, is Aylia Özdan helping someone else on purpose by spending some of her spare time.

Scenario 2 (a) *Mr. Beck is 70 years old and interested in playing Backgammon regularly. None of his acquaintances is playing it, and he doesn't know any other person who might play it.* (b) *Mr. Miller plays regularly Poker with his buddies on Friday evening and is always looking for new participants who are interested in it.* (c) *Mr. Novak wants to play Skat with a friend and they are looking for a third player.*

In this scenario Mr. Beck is looking for someone who is playing Backgammon. Because there is nobody else in the system who is interested in it, no exact match is possible. But there are other requests stored in the system, like Skat and Poker, the system could offer instead as similar matches. The implication is, if one is interested in playing Backgammon, one could also be interested in playing other games. Here, similarity means finding someone with similar interests. At the level of a "parlour game" all these requests are similar. So they should appear in the result list as possible matches.

4 REQUIREMENTS

In chapter 3 we described two different kinds of scenarios where matchmaking is either a best fit or a similarity match. From these scenarios 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. Here, 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. 2.2) matches users with similar interests it does not handle fitting of offers with requests. Classads (Sec. 2.1), on the other hand, support these different roles but only provides the data structure and no matching algorithm. The fuzzy multiset approach (Sec. 2.3) also supports the different roles of requester and provider as long as all parameters can be expressed as 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. If one soft constraint can't be satisfied, the activity will still be available as a possible match. 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 (Backgammon) is an example where a lot of soft constraints exist: time, place, day of week, and even the activity itself. Whereas the examples of scenario 1 have a lot of hard constraints, like time, place, bus line, 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 datastructure 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. Via user roles these differences can be modelled. 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 activites.

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 default role represents those users who are searching for help or looking for company. The difference in helping is taken into account by another two roles. That is, volunteers and 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. In a third role are those users represented doing favours. The difference to volunteers is, they don't help out regularly and they 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. For matchmaking based on similarity different sources of knowledge are suitable. That is, 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. Then, the matching service can offer similar activities by evaluating it (e. g. by taxonomic relationships). Suppose, someone searches for Backgammon, but there is no direct match (as the situation is in scenario 2). The matchmaking service can offer Skat and Poker instead and ignore other available activities. 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. The initial background knowledge would be more dynamic and converge 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 everytime 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 withhold 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 more 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 [9] 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? [2] 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. Moreovere, 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 [3] 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 [3]. 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 play Backgammon and issues a search. In the profile aren't defined any preferences, like hard constraints. Further assume no exact match is possible, but there are two other activites stored (Skat and Poker), as the situation in scenario 2. As a result, the best matches are Poker and Skat. 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

Everytime 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 modus. 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. Utilising a stalled request one can find a match that best fits over a period of time while an immediate request matches the best fit of the current available activities.

Classads [9] and the fuzzy multiset approach [3] 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. Whereas, in R-U-In? [2] you can search for and post activities. Activities are stored in a so-called activity groups reposititory. The difference here 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 1 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 is 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 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

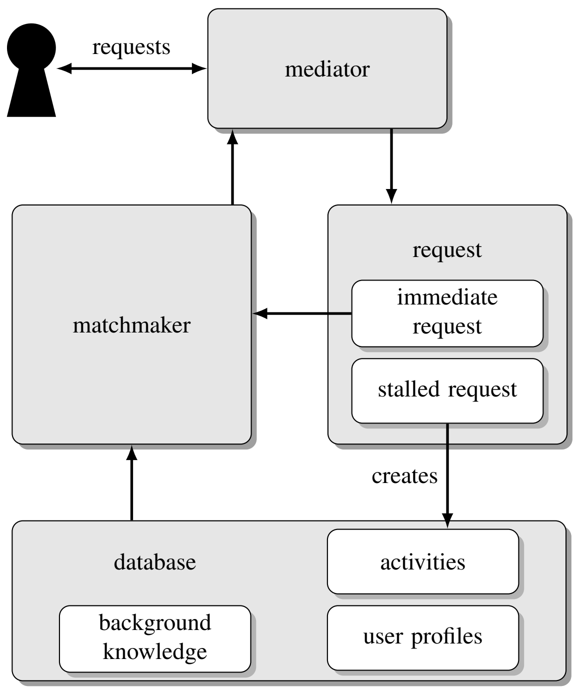

Figure 1. Components of the matching framework and their interaction. The matchmaker is the key component of the system. It uses the underlying database for a matching and propagates the results to the mediator.

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 like 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 mapping of fittings can be modelled in such a way that the resulting scale corresponds to a similarity mapping. So that 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:

$$\mathit{other.gender} = \mathit{female} \quad \wedge \quad \mathit{other.age} \geq 30 \tag{1}$$

where *other* is a reference to a potential activity partner (similar to [9]). Consider the request of Mrs. Peters finding someone assisting her in riding the public bus as activity a_1:

$$\mathit{requires}(a_1, \mathit{assistance}) \tag{2}$$

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

$$\mathit{provides}(a_2, \mathit{assistance}) \tag{3}$$

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

$$\mathit{likes}(\mathit{other}, \mathit{backgammon}) \simeq \mathit{likes}(\mathit{other}, \mathit{skat}) \tag{4}$$

Using constraints, time and loaction 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 $\mathit{requires}$ and $\mathit{provides}$ 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 being 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 normalized 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 obtained by a sum

$$V \;=\; \sum_{c \in C} s_c \cdot w_c \tag{5}$$

where s_c represents the normalized 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 implicit. 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. Manually constraining the search should have higher priority as constraints in profiles. For this reason, different knowledge has different priority. Information given in profiles have 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 $ignore(cardgames)$ in his profile the constraint specified and searches for Backgammon. Skat, Poker, and chess are in the system as available activities. Because Backgammon isn't available, the only similar activities are Skat, Poker, and chess. The matching service just offers chess as an alternative activity and discards the card games Skat and Poker, because they're on the ignore list. Now suppose, the same user initiates an explicit request for Skat. The request has higher priority as the constraint in the profile and overwrites it. This approach allows users to find still activities explicit, even when the profile states otherwise, by overwriting constraints.

Further, it is also important not to treat unmatched constraints as fails because of missing information about the feasibility on the other side. Assume Zoey wants to attend a concert, but needs someone with a car to go there. So the car is a requirement that can be modelled as a (hard) constraint: $requires(car)$. Her neighbour Tim also wants to go to the concert. However, he doesn't mention in his stalled activity that he's going to drive with his own car. The problem here is, Zoey 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 Zoey about the possible match and propose her to contact Tim to check, if the activity can be matched anyway. After contacting Tim, Zoey is able to go with Tim to the concert by car.

Missing information can be treated as *wild cards* which match everything. The matchmaker doesn't know if Tim possesses a car, but the requirement is assumed to be fulfilled. However, the activity is marked as *uncertain* until Tim 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 CONCLUSION AND FUTURE WORK

In this paper we proposed a framework for matchmaking similar activities. It is part of a larger web-application called EMN-MOVES. The target group of the system are older people and the overall goal is to improve their mobility and their social life. We described two different kinds of scenarios the system might be confronted with to derive the requirements of the framework. We have evaluated the requirements against existing approaches and concluded none of these can fully support our needs for a platform based on neighbourly help.

The presented framework will be the starting point for the development of a general framework for matchmaking. Currently, we are designing an algorithm which allows to calculate both similarity matches and best fits and incorporates a goodness criterion for ranking the results.

ACKNOWLEDGEMENTS

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] Serge Abiteboul, ‘Querying Semi-Structured Data’, in *Database Theory - ICDT ’97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, pp. 1–18, (1997).

[2] Nilanjan Banerjee, Dipanjan Chakraborty, Koustuv Dasgupta, Sumit Mittal, Seema Nagar, 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*, pp. 222–231. IEEE, (2009).

[3] Giuseppe Fenza, Vincenzo Loia, and Sabrina Senatore, ‘A hybrid approach to semantic web services matchmaking’, *International Journal of Approximate Reasoning*, **48**(3), 808–828, (2008).

[4] Dedre Gentner and Arthur B. Markman, ‘Defining structural similarity’, *The Journal of Cognitive Science*, **6**, 1–20, (2006).

[5] Robert L. Goldstone and Ji Yun Son, ‘Similarity’, *Psychological Review*, **100**, 254–278, (2004).

[6] Javier González-Castillo, David Trastour, and Claudio Bartolini, ‘Description Logics for Matchmaking of Services’, in *IN PROCEEDINGS OF THE KI-2001 WORKSHOP ON APPLICATIONS OF DESCRIPTION LOGICS*, (2001).

[7] Ian Horrocks and Peter Patel-Schneider, ‘Reducing OWL Entailment to Description Logic Satisfiability’, in *The Semantic Web - ISWC 2003*, eds., Dieter Fensel, Katia Sycara, and John Mylopoulos, volume 2870 of *Lecture Notes in Computer Science*, 17–29, Springer Berlin / Heidelberg, (2003).

[8] Sadaaki Miyamoto, ‘Information clustering based on fuzzy multisets’, *Inf. Process. Manage.*, **39**(2), 195–213, (March 2003).

[9] Rajesh Raman, Miron Livny, and Marvin Solomon, ‘Matchmaking: Distributed resource management for high throughput computing’, in *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pp. 140–146. IEEE, (1998).

[10] Ute Schmid, Michael Siebers, Johannes Folger, Simone Schineller, Dominik Seuss, Marius Raab, Claus-Christian Carbon, and Stella Faerber, ‘A cognitive model for predicting aesthetical judgements as similarity to dynamic prototypes’.

[11] D. Shaw, P.E. Newson, P.W. O’Kelley, and W.B. Fulton. Social matching of game players on-line, 2005.

[12] *Online matchmaking*, eds., Monica T. Whitty, Andrea J. Baker, and James A. Inman, Basingstoke, 2007.