Exploiting User Feedback to Improve Semantic Web Service Discovery

Anna Averbakh, Daniel Krause, Dimitrios Skoutas

L3S Research Center
Hannover, Germany
{averbakh,krause,skoutas}@l3s.de

Abstract. State-of-the-art discovery of Semantic Web services is based on hybrid algorithms that combine semantic and syntactic matchmaking. These approaches are purely based on similarity measures between parameters of a service request and available service descriptions, which, however, fail to completely capture the actual functionality of the service or the quality of the results returned by it. On the other hand, with the advent of Web 2.0, active user participation and collaboration has become an increasingly popular trend. Users often rate or group relevant items, thus providing valuable information that can be taken into account to further improve the accuracy of search results. In this paper, we tackle this issue, by proposing a method that combines multiple matching criteria with user feedback to further improve the results of the matchmaker. We extend a previously proposed dominance-based approach for service discovery, and describe how user feedback is incorporated in the matchmaking process. We evaluate the performance of our approach using a publicly available collection of OWL-S services.

1 Introduction

Web services have emerged as a key technology for implementing Service Oriented Architectures, aiming at providing interoperability among heterogeneous systems and integrating inter-organization applications. At the same time, users increasingly use the Web to search not only for pages or other multimedia resources, but also to find services for fulfilling a given task. For example, a service typically either returns some information to the user, such as a weather forecast for a given location and time period, or it performs some task, such as flight booking for a given date, departure and destination. Hence, as both the user needs and the number of available services and service providers increases, improving the effectiveness and accuracy of Web service discovery mechanisms becomes a crucial issue.

A Web service description is a document written in a formal, standardized language, providing information about what the service does and how it can be used. Given a service request, expressed as the description of a desired service, the task of matchmaking refers to identifying and selecting from a service repository those services whose description closely matches the request, under

one or more matching criteria. Typically, the matchmaking process is based on computing a degree of match between the parameters of the requested and advertised service, which may include both functional parameters (i.e., inputs, outputs, pre-conditions, and effects), as well as non-functional parameters (i.e., QoS attributes, such as price, execution time, availability).

Several approaches, reviewed in Section 2, exist for this task, ranging from methods relying on techniques commonly used in Information Retrieval [1], to methods employing logic-based matching [2]. Also, hybrid methods have been proposed [3], and, more recently, methods that combine multiple matching criteria simultaneously [4]. Nevertheless, all these works rely solely on the parameters involved in the service descriptions, thus facing an important drawback. A service description may not always capture completely and accurately all the aspects related to the functionality and usage of the service. Additional information which may be useful for determining how appropriate a service is for a given request, often remains implicit, not being encoded in the formal description. Moreover, different services may be more relevant to a specific request for different users and in different contexts.

On the other hand, with the advent of Web 2.0, active user participation and collaboration has become an increasingly popular trend. Users provide valuable information by tagging, rating, or grouping similar resources, such as bookmarks (e.g., Del.icio.us), music (e.g., Last.fm) or videos (e.g., YouTube). Essentially, these activities allow collecting user feedback, which can then be exploited to enhance the accuracy and effectiveness of the search [5].

In this paper, we propose a method for leveraging user feedback to improve the results of the service discovery process. Given a service request, the matchmaker searches the repository for available services and returns a ranked list of candidate matches. Then, the system allows the user posing the query to rate any of these matches, indicating how relevant or appropriate they are for this request. The provided ratings are stored in the system for future use, when the same or a similar request is issued. Designing intuitive, easy-to-use user interfaces, can help the process of collecting user feedback. In this work, we do not deal with this issue; instead, our focus is on how the collected feedback is processed and integrated in the matchmaking process to improve the results of subsequent searches. Notice, that it is also possible to collect user feedback automatically, assuming that the system can track which service(s) the user actually used; however, this information would typically be incomplete, since not all relevant services are used.

Our main contributions are summarized below.

- We consider the problem of employing user feedback to improve the quality of search for Semantic Web services.
- We propose a method for processing user feedback and incorporating it in the matchmaking process.
- We experimentally evaluate our approach on a publicly available collection of Semantic Web services, applying standard Information Retrieval evaluation metrics.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents our architecture for incorporating user feedback in the service discovery process. Section 4 describes the basic matchmaking approach, employing multiple matching criteria. Applying user feedback to improve the search results is presented in Section 5. Finally, Section 6 presents our experimental results, while Section 7 concludes the paper.

2 Related Work

WSDL and UDDI are existing industry standards for describing and discovering Web services. However, their focus lies on specifying the structure of the service interfaces and of the exchanged messages. Thus, they address the discovery problem relying on structural, keyword-based matching, which limits their search capabilities. Other earlier works have also focused on applying Information Retrieval techniques to the service discovery problem. For example, the work presented in [1] deals with similarity search for Web services, using a clustering algorithm to group names of parameters into semantically meaningful concepts, which are then used to determine the similarity between input/output parameters. An online search engine for Web services is seekda[1], which crawls and indexes WSDL files from the Web. It allows users to search for services by entering keywords, by using tag clouds, or by browsing using different facets, such as the country of the service provider, the most often used services or the most recently found ones.

To deal with the shortcomings of keyword search, several approaches have been proposed for exploiting ontologies to semantically enhance the service descriptions (WSDL-S [6], OWL-S [7], WSMO [8]). These so-called Semantic Web services can better capture and disambiguate the service functionality, allowing for formal, logic-based matchmaking. Essentially, a logic reasoner is employed to infer subsumption relationships between requested and provided service parameters [2, 9]. Along this line, several matching algorithms assess the similarity between requested and offered inputs and outputs by comparing the positions of the corresponding classes in the associated domain ontology [10–12]. Similarly, the work in [13] semantically matches requested and offered parameters, modeling the matchmaking problem as one of matching bipartite graphs. In [14], OWL-S services are matched using a similarity measure for OWL objects, which is based on the ratio of common RDF triples in their descriptions. An approach for incorporating OWL-S service descriptions into UDDI is presented in [15], focusing also on the efficiency of the discovery process. Efficient matchmaking and ranked retrieval of services is also studied in [16].

Given that logic-based matching can often be too rigid, hybrid approaches have also been proposed. In an earlier work [17], the need for employing many types of matching has been discussed, proposing the integration of multiple external matching services to a UDDI registry. The selection of the external matching

[1] http://seekda.com/

service to be used is based on specified policies, e.g., selecting the first available, or the most successful. If more than one matching services are invoked, again the system policies specify whether the union or the intersection of the results should be returned. OWLS-MX [3] and WSMO-MX [18] are hybrid matchmakers for OWL-S and WSMO services, respectively. More recently, an approach for simultaneously combining multiple matching criteria has been proposed [4].

On the other hand, some approaches already exist about involving the user in the process of service discovery. Ontologies and user profiles are employed in [19], which then uses techniques like query expansion or relaxation to better satisfy user requests. The work presented in [20] focuses on QoS-based Web service discovery, proposing a reputation-enhanced model. A reputation manager assigns reputation scores to the services based on user feedback regarding their performance. Then, a discovery agent uses the reputation scores for service matching, ranking and selection. The application of user preferences, expressed in the form of soft constraints, to Web service selection is considered in [21], focusing on the optimization of preference queries. The approach in [22] uses utility functions to model service configurations and associated user preferences for optimal service selection. In [1], different types of similarity for service parameters are combined using a linear function, with manually assigned weights. Learning the weights from user feedback is proposed, but it is left as an open issue for future work.

Collecting and exploiting user feedback as a form of interaction between a user and an application is a key concept of Web 2.0 [23]. Users are more than ever before willing to actively contribute by tagging, rating or commenting any kind of content or offered service – even better, their feedback is very valuable to improve search and retrieval algorithms [24, 25]. In this work we employ a simple user feedback model, focusing on how this feedback can be exploited in Semantic Web service discovery, to improve the quality and accuracy of the search results. More fine-grained forms of user feedback, as well as exploiting additional information from social networks of users [26] are considered as possible extensions of our approach.

In a different line of research, relevance feedback has been extensively considered in Information Retrieval [27]. The main idea is that the user issues a query, the system returns an initial set of results, and the user marks some of the returned documents as relevant or non-relevant; then, based on this user feedback, the system revises the set of retrieved results to better capture the user's information need. However, approaches in this area typically rely on the vector space model and term re-weighting, and, hence, they are not suitable for service matchmaking.

3 Architecture

Typical service matchmaking systems are based on a unidirectional information flow. First, an application that needs a specific Web Service to perform a task creates a service request, containing the requirements that a service should fulfil. This service request is then delivered to a matchmaking component that utilizes

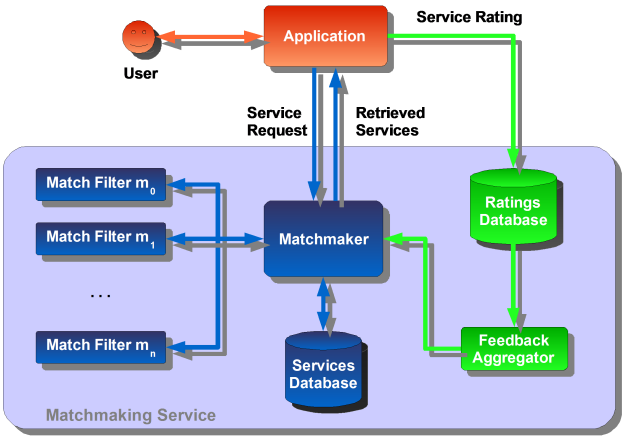


Fig. 1. Matchmaking service with feedback component

one or more match filters to retrieve the best-matching services from a repository of Semantic Web service descriptions. These services are finally returned to the application which invoked the matchmaker. The drawback in this scenario is that if a service is not appropriate or sufficient for any reason to perform the original task, the application has no option to inform the matchmaker about the inappropriateness of this match result.

Hence, our matchmaking architecture is extended by a feedback loop, as illustrated in Figure 1, enabling the matchmaking mechanism to use previously provided user feedback in order to improve the quality of the retrieved results.

Enabling this feedback loop, relies on the assumption that the application users can assess the quality of retrieved Web services. This is a common principle in Web 2.0 applications, where users can rate available resources. One possibility is that users can rate services explicitly. If it is not possible or easy for the users to rate services directly, the application can still infer implicit ratings for a service through user behavior. For example, if an applications uses services to generate music recommendations, then users can be asked whether they consider the given recommendations appropriate. Based on the assumption that services delivering high quality recommendations are better matches for this task, the application can infer the relevance of a service, and pass this information as a user rating to the matchmaking service.

The user ratings are stored in an RDF triple store (e.g., SESAME [28]). As user ratings refer to a given service request, each `Rating` instance contains the user who performed the rating, the service request, the rated service, and finally a rating score that ranges from 0 to 1 (with higher scores denoting higher rating). For example, a rating from Bob about a request X and a service Y would be stored as:

```
<r:Rating>
  <foaf:Person rdf:about="#bob"/>
  <r:Request rdf:about="#requestX"/>
  <r:Service rdf:about="#serviceY"/>
  <r:Score rdf:datatype="&xsd;double">0.90</r:score>
</r:Rating>
```

This RDF database, which contains the user feedback in form of ratings, is exploited by the user feedback component. This component aggregates previous ratings provided by different users, to determine the relevance between a service request and an actual service.

Then, given a service request, the matchmaker component combines the relevance score from the feedback component with the similarity scores calculated by the match filter(s) to assess the degree of match for each available service, and returns a ranked list of match results to the application.

4 Service Matchmaking

In this section we describe the basic service matchmaking and ranking process, without taking into account user feedback. For this task, we adopt the approach from [4], because, as shown in Section 5, it allows us to integrate user feedback in a more flexible and seamless way. In the following, we give a brief overview of how the matchmaking and ranking of services is performed.

For simplicity, we focus on input and output parameters, annotated by ontological concepts, but other types of parameters can be handled similarly. Let R be a service request with a set of input and output parameters, denoted by R_{IN} and R_{OUT}, respectively. We use $R.p_j$ to refer to the j-th input parameter, where $p_j \in R_{IN}$ (similarly for outputs). Also, assume an advertised service S with inputs and outputs S_{IN} and S_{OUT}, respectively. Note that S can be a match to R, even when the cardinalities of their parameter sets differ, i.e., when a service advertisement requires less inputs or produces more outputs than requested.

The matchmaking process applies one or more matching functions to assess the degree of match between requested and offered parameters. Each matching function produces a score in the range $[0, 1]$, where 1 indicates a perfect match, while 0 indicates no match. Typical examples are the matching functions provided by the OWLS-MX service matchmaker [3]. These comprise a purely logic-based match (M0), as well as hybrid matches based on string similarity measures, namely loss-of-information (M1), extended Jaccard similarity coefficient (M2), cosine similarity (M3), and Jensen-Shannon information divergence based similarity (M4). Given a request R, a service S, and a matching function m_i, the *match instance* of S with respect to R is defined as a vector s_i such that

$$s_i[j] = \begin{cases} \max\limits_{p_k \in S_{IN}} \{m_i(S.p_k, R.p_j)\}, & \forall j : p_j \in R_{IN} \\ \max\limits_{p_k \in S_{OUT}} \{m_i(S.p_k, R.p_j)\}, & \forall j : p_j \in R_{OUT} \end{cases} \quad (1)$$

Match Filter	Book	Price
M0	0.88	1.00
M1	0.93	1.00
M2	0.69	1.00
M3	0.72	1.00
M4	0.93	1.00

Table 1. Example of the match object for the request `book_price_service.owls` and the service `novel_price_service.owls`

The match instance s_i has a total of $d = |R_{IN}| + |R_{OUT}|$ entries that correspond to the input and output parameters of the request. Intuitively, each s_i entry quantifies how well the corresponding parameter of the request R is matched by the advertisement S, under the matching criterion m_i. Clearly, an input (output) parameter of R can only match with an input (output) parameter of S.

Let $\mathcal{M}$ be a set of matching functions. Given a request R and an advertisement S, each $m_i \in \mathcal{M}$ results in a distinct match instance. We refer to the set of instances as the *match object* of the service S. In the following, and in the context of a specific request R, we use the terms service and match object interchangeably, denoted by the same uppercase letter (e.g., S). On the other hand we reserve lowercase letters for match instances of the corresponding service (e.g., s_1, s_2, etc.). The notation $s_i \in S$ implies that the match instance s_i corresponds to the service S. Hence, a match object represents the result of the match between a service S and a request R, with each contained match instance corresponding to the result of a different match function.

As a concrete example, consider a request `book_price_service.owls`, with input `Book` and output `Price`, and a service `novel_price_service.owls`, with input `Novel` and output `Price`, matched applying the five aforementioned functions M0–M4 of the OWLS-MX matchmaker. The resulting match object is shown in Table 1, comprising an exact match for the outputs, while the degree of match between `Book` and `Novel` varies based on the similarity measure used.

Next, we describe how services are ranked based on their match objects. Let $\mathcal{I}$ be the set of all match instances of all services. Given two instances $u, v \in \mathcal{I}$, we say that u *dominates* v, denoted by $u \succ v$, iff u has a higher or equal degree of match in all parameters and a strictly higher degree of match in at least one parameter compared to v. Formally

$$u \succ v \quad \Leftrightarrow \quad \forall i \; u[i] \geq v[i] \wedge \exists j \; u[j] > v[j] \tag{2}$$

If u is neither dominated by nor dominates v, then u and v are incomparable.

Given this dominance relationship between match instances, we proceed with defining *dominance scores* that are used to rank the available service descriptions with respect to a given service request. Intuitively, a service should be ranked highly in the list if

– its instances are dominated by as few other instances as possible, and

– its instances dominate as many other instances as possible.

To satisfy these requirements, we formally define the following dominance scores, used to rank the search results for a service request.

Given a match instance u, we define the *dominated score* of u as

$$u.dds = \frac{1}{|\mathcal{M}|} \sum_{V \neq U} \sum_{v \in V} |v \succ u| \tag{3}$$

where $|u \succ v|$ is 1 if $u \succ v$ and 0 otherwise. Hence, $u.dds$ accounts for the instances that dominate u. Then, the dominated score of a service U is defined as the (possibly weighted) average of the dominated scores of its instances:

$$U.dds = \frac{1}{|\mathcal{M}|} \sum_{u \in U} u.dds \tag{4}$$

The dominated score of a service indicates the average number of services that dominate it, i.e., a *lower* dominated score indicates a better match result.

Next, we look at the instances that a given instance dominates. Formally, given a match instance u, we define the *dominating score* of u as

$$u.dgs = \frac{1}{|\mathcal{M}|} \sum_{V \neq U} \sum_{v \in V} |u \succ v| \tag{5}$$

Similarly to the case above, the dominating score of a service U is then defined as the (possibly weighted) average of the dominating scores of its instances:

$$U.dgs = \frac{1}{|\mathcal{M}|} \sum_{u \in U} u.dgs \tag{6}$$

The dominating score of a service indicates the average number of services that it dominates, i.e., a *higher* dominating score indicates a better match result.

Finally, we define the *dominance score* of match instances and services, to combine both of the aforementioned criteria. In particular, the dominance score of a match instance u is defined as

$$u.ds = u.dgs - \lambda \cdot u.dds \tag{7}$$

where the parameter λ is a scaling factor. This promotes u for each instance it dominates, while penalizing it for each instance that dominates it. Then, the dominance score of a service U is defined as the (possibly weighted) average of the dominance scores of its instances:

$$U.ds = \frac{1}{M} \sum_{u \in U} u.ds \tag{8}$$

The ranking process comprises computing the aforementioned scores for each service, and then sorting the services in descending order of their dominance score. Efficient algorithms for this computation can be found in [4].

5 Incorporating User Feedback

In this section, we present our approach for processing user feedback and incorporating it in the dominance-based matchmaking and ranking method described in Section 4.

Our approach is based on the assumption that the system collects feedback from the users by allowing them to rate how appropriate the retrieved services are with respect to their request (see feedback component in Section 3). Assume that the collected user ratings are stored as a set $\mathcal{T} \subseteq \mathcal{U} \times \mathcal{R} \times \mathcal{S} \times F$ in the Ratings Database, where $\mathcal{U}$ is the set of all users that have provided a rating, $\mathcal{R}$ is the set of all previous service requests stored in the system, $\mathcal{S}$ is the set of all the available Semantic Web service descriptions in the repository, and $F \in [0, 1]$ denotes the user rating, i.e., how relevant a particular service was considered with respect to a given request (with higher values representing higher relevance). Thus, a tuple $T = (U, R, S, f) \in \mathcal{T}$ denotes that a user U considers the service $S \in \mathcal{S}$ to be relevant for the request $R \in \mathcal{R}$ with a score f.

To aggregate the ratings from different users into a single feedback score, different approaches can be used. For example, [29] employs techniques to identify and filter out ratings from spam users, while [30] proposes the aging of feedback ratings, considering the more recent ratings as more relevant. It is also possible to weight differently the ratings of different users, assigning, for example, higher weights to ratings provided previously by the same user as the one currently issuing the request, or by users that are assumed to be closely related to him/her, e.g., by explicitly being included in his/her social network or being automatically selected by the system through techniques such as collaborative filtering or clustering. However, as the discussion about an optimal aggregation strategy for user ratings is orthogonal to our main focus in this paper, without loss of generality we consider in the following all the available user ratings as equally important, and we calculate the feedback value as the average of all user ratings of the corresponding service. Hence, the feedback score fb between a service request $R \in \mathcal{R}$ and a service advertisement $S \in \mathcal{S}$ is calculated as:

$$fb(R, S) = \frac{\sum_{(U,R,S,f) \in \mathcal{T}} f}{|\{(U, R, S, f) \in T\}|} \tag{9}$$

However, it may occur that for a given pair of a request R and a service S, no ratings (U, R, S, f) exist in the database. This may be because the request R is new, or because the service S has been recently added to the database and therefore has been only rated for a few requests. Moreover, even if some ratings exist, they may be sparse and hence not provide sufficiently reliable information for feedback. In these cases, Equation (9) is not appropriate for determining the feedback information for the pair (R, S). To address this issue, we generalize this method to consider not only those ratings that are directly assigned to the current service requests R, but also user ratings that are assigned to requests that are similar to R. Let $SIM(R)$ denote the set of requests which are considered to be similar to R. Then, the feedback can be calculated as:

$$fb(R,S) = \frac{\sum_{(U,Q,S,f)\in\mathcal{T}:Q\in SIM(R)} f * sim(R,Q)}{|\{(U,Q,S,f)\in\mathcal{T} : Q\in SIM(R)\}|} \quad (10)$$

In Equation (10), $sim(R,Q)$ is the match instance of Q with respect to R, calculated by a matching function m_i, as discussed in Section 4. Notice that $sim(R,Q)$ is a vector of size equal to the number of parameters of R, hence in this case $fb(R,S)$ is also such a vector, i.e., similar to a match instance. Also, Equation (9) can be derived as a special case of Equation (10), by considering $SIM(R) = \{R\}$. By weighting the given feedback by the similarity between the requests, we ensure that feedback from requests which are more similar to the considered one, is taken more into account.

A question that arises is how to select the similar requests for a given request R, i.e., how to determine the set $SIM(R)$. This choice involves a trade-off. Selecting a larger number of similar queries, allows the use of more sources of information for feedback; however, if the similarity between the original request and the selected ones is not high enough, then the information from this feedback is also not highly appropriate, and may eventually introduce noise in the results. On the other hand, setting a very strict criterion for selecting similar queries, reduces the chance of finding enough feedback information. As a solution to this trade-off, we use a top-k query with constraints: given a request R, we select the top-k most similar requests from the database, given that the values of their match instances are above a specified threshold.

The process described above results in a *feedback instance* $fb(R,S)$ for the given request R and a service S. The next step is to integrate this instance to the match object of the service S, comprising the other instances obtained by the different matching functions m_i. We investigate two different strategies for this purpose:

1. *Feedback instance as an additional match instance.* In this case we add the feedback information to the match object of the service as an additional instance (combined with the average of the previous values). That is, this method treats the feedback mechanism as an extra matchmaking function.
2. *Feedback instance integrated with match instances.* In this case we update the values of the match instances by adding the values of the feedback instance. That is, this method adjusts the results of the matchmaking functions applying the feedback information.

As a concrete example, consider the match object presented in Table 1. Assume that the feedback instance for the pair (`book_price_service.owls`, `novel_price_service.owls`) is fb = [0.77 1.00]. Then this match object will be modified as shown in Table 2.

(a) Method 1

Match Filter	Book	Price
M0	0.88	1.00
M1	0.93	1.00
M2	0.69	1.00
M3	0.72	1.00
M4	0.93	1.00
AVG(M_i)+FB	1.60	2.00

(b) Method 2

Match Filter	Book	Price
M0+FB	1.65	2.00
M1+FB	1.70	2.00
M2+FB	1.46	2.00
M3+FB	1.49	2.00
M4+FB	1.70	2.00

Table 2. Example of the match object for the request `book_price_service.owls` and the service `novel_price_service.owls` updated using feedback information

6 Experimental Evaluation

In this section, we evaluate the quality of our feedback-based matchmaking approach in comparison to state-of-the-art matchmaking algorithms. In Section 6.1, we discuss implementation issues and the data collections we have used. The evaluation metrics and the experimental results are then reported in Section 6.2.

6.1 Experimental Setup

We have implemented the feedback-based matchmaking and ranking process described in Sections 4 and 5. The implementation utilizes the OWLS-MX service matchmaker [3], to process service requests and advertisements described in OWL-S, and to compute the pairwise similarities between parameters. In particular, OWLS-MX provides five different matching filters. The first performs a purely logic-based match (M0), characterising the result as exact, plug-in, subsumes, or subsumed-by. The other four perform hybrid match, combining the semantic-based matchmaking with the following measures: loss-of-information (M1), extended Jaccard similarity coefficient (M2), cosine similarity (M3), and Jensen-Shannon information divergence based similarity (M4). Notice, that for each pair (R, S) of a service request and service advertisement, OWLS-MX applies one of the filters M0–M4, and calculates a single score denoting the degree of match between R and S. We have modified this functionality to get all the individual degrees of match between the compared parameters of R and S (i.e., a vector); also, we have applied for each pair (R, S) all the similarity measures M0–M4, to get the individual match instances, as described in Section 4. Finally, our implementation includes also the process described in Section 5 for processing and using the available feedback information.

For our experiments, we have used the publicly available service retrieval test collection OWLS-TC v2[2]. This collection comes in two versions, an original one containing 576 services, and an extended one, containing 1007 services. To

[2] This collection is available at `http://projects.semwebcentral.org/projects/owls-tc/`. Before running the experiments we have fixed some typos that prevented some services from being processed and/or retrieved.

Collection	# of requests	# of services	# of rel. services per req. (average)
OWL-S TC I	28	576	15.2
OWL-S TC II	28	1007	25.4

Table 3. Characteristics of the test collections

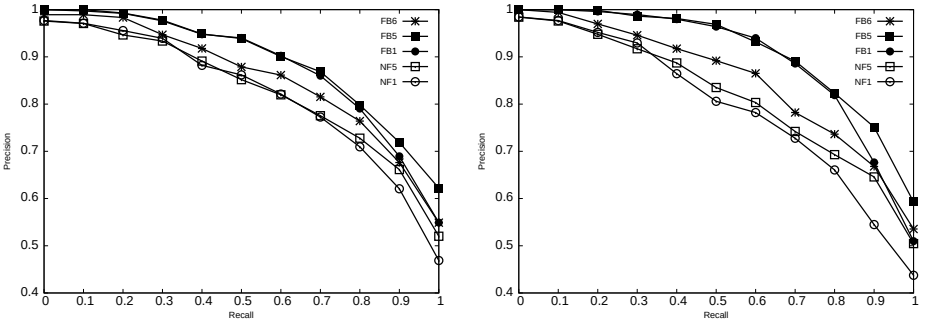
better assess the performance of our method, we have conducted our experiments on both versions, denoted in the following as OWLS-TC I and OWLS-TC II, respectively. The contained service descriptions are based on real-world Web services, retrieved mainly from public IBM UDDI registries, covering 7 different domains, such as economy, education, and travel. Also, the collection comprises a set of 28 sample requests. Notice that the extended version of the collection comprises one extra request, namely `EBookOrder1.owls`; however, in our experiments, we have excluded this request, so that in both cases the set of queries used for the evaluation is the same. For each request, a relevance set is provided, i.e., the list of services that are considered relevant to this request, based on human judgement. The characteristics of the two data sets are summarized in Table 3.

To evaluate our feedback-based mechanism, there needs to be, for each request, at least one similar request for which some services have been rated as relevant. As this was not the case with the original data set, due to the small number of provided requests, we have extended both of the aforementioned collections by creating a similar query for each of the 28 original ones. This was done by selecting a request, then selecting one or more of its input and/or output parameters, and replacing its associated class in the ontology with one that is a superclass, subclass or sibling. Then, for each of these newly created queries, some of the services in the collection were rated as relevant. To simplify this task, we have restricted our experimental study in binary ratings, i.e., the value of the user rating was either 1 or 0, based on whether the user considered the service to be relevant to the request or not. The new queries and the ratings, provided in the form of corresponding relevance sets, are made available for further use at: `http://www.l3s.de/~krause/collection.tar.gz`.

6.2 Experimental Results

In the following, we evaluate the performance of our approach, including both strategies described in Section 5. For this purpose, we compare the retrieved results to the ones produced without taking user feedback into consideration. In particular, we have implemented and compared the following 5 methods:

- *NF1*: No feedback is used; one match instance per service is considered. The values of the match instance are the degrees of match between the request and service parameters, computed applying the Jensen-Shannon similarity measure, i.e., the filter M4 from OWLS-MX, which is shown in [3] to slightly outperform the other measures.



(a) OWLS-TC I (b) OWLS-TC II

Fig. 2. Precision-Recall curve for the OWLS test collections

- *NF5*: No feedback is used; five match instances per service are considered. The values of the match instances are the degrees of match between the request and service parameters computed by the filters M0–M4 of OWLS-MX.
- *FB1*: Feedback is used; one match instance per service is considered. The values of the match instance are the sum of the degrees of match between the request and service parameters computed by M4 and the feedback values calculated by Equation (10).
- *FB5*: Feedback is used; five match instances per service are considered. The value of each match instance is the sum of the degrees of match between the request and service parameters computed by one of the measures M0–M4 and the feedback values calculated by Equation (10).
- *FB6*: Feedback is used; six match instances per service are considered. The values of the first five match instances are the degrees of match between the request and service parameters computed by the filters M0–M4. The values of the sixth match instance are computed as the averages of the previous ones plus the feedback values calculated by Equation (10). Notice, that the reason for using also the average values of the initial instances, instead of only the feedback values, is mainly to avoid penalizing services that constitute good matches but have not been rated by users.

To measure the effectiveness of the compared approaches, we apply the following standard IR evaluation measures [27]:

- *Interpolated Recall-Precision Averages*: measures precision, i.e., percent of retrieved items that are relevant, at various recall levels, i.e., after a certain percentage of all the relevant items have been retrieved.
- *Mean Average Precision (MAP)*: average of precision values calculated after each relevant item is retrieved.

(a) OWLS-TC I

Method	MAP	R-prec	bpref	R-rank	P@5	P@10	P@15	P@20
FB6	0.8427	0.7772	0.8206	0.9762	0.9214	0.8357	0.7690	0.6589
FB5	0.8836	0.7884	0.8600	1.0000	0.9714	0.8857	0.7952	0.6696
FB1	0.8764	0.7962	0.8486	1.0000	0.9786	0.8786	0.7929	0.6625
NF5	0.8084	0.7543	0.7874	0.9405	0.9071	0.7964	0.7500	0.6393
NF1	0.8027	0.7503	0.7796	0.9405	0.9214	0.8143	0.7357	0.6357

(b) OWLS-TC II

Method	MAP	R-prec	bpref	R-rank	P@5	P@10	P@15	P@20
FB6	0.8426	0.7652	0.8176	1.0000	0.9714	0.8964	0.8476	0.7875
FB5	0.9090	0.8242	0.8896	1.0000	0.9857	0.9679	0.9214	0.8536
FB1	0.8960	0.8024	0.8689	1.0000	0.9857	0.9607	0.9167	0.8411
NF5	0.8007	0.7388	0.7792	0.9643	0.9429	0.8607	0.8119	0.7536
NF1	0.7786	0.7045	0.7499	0.9643	0.9357	0.8607	0.7976	0.7268

Table 4. IR metrics for the OWLS test collections

- *R-Precision (R-prec)*: measures precision after all relevant items have been retrieved.
- *bpref*: measures the number of times judged non-relevant items are retrieved before relevant ones.
- *Reciprocal Rank (R-rank)*: measures (the inverse of) the rank of the top relevant item.
- *Precision at N (P@N)*: measures the precision after N items have been retrieved.

Figure 2 plots the precision-recall curves for the 5 compared methods, for both considered test collections. Overall, the main observation is that the feedback-aware methods clearly outperform the other two ones in both test collections. The best overall method in both collections is $FB5$, because it provides two advantages: a) it utilizes user feedback, and b) it combines all the available similarity measures for matchmaking service parameters. The method $FB1$, which combines feedback information with the Jensen-Shannon hybrid filter, also demonstrates a very high accuracy. The method $FB6$, which treats the feedback information as an additional match instance, achieves lower precision, but it still outperforms the non-feedback methods. This behavior is due to the fact that although feedback is utilized, its impact is lower since it is not considered for the 5 original match instances, but only as an extra instance. Regarding $NF5$ and $NF1$, the former exhibits better performance, which is expected as it combines multiple similarity measures. Another interesting observation is that $FB5$ and $FB1$ follow the same trend as $NF5$ and $NF1$, respectively, which are their non-feedback counterparts, however having considerably higher precision values at all recall levels. Finally, for the collection OWLS-TC II, which comprises an almost double number of services, the trends are the same as before, but with the differences between the feedback-aware and the non-feedback methods being even more noticeable. Another interesting observation in this case is that after the recall level 0.8 the precision of $FB1$ drops much faster than that of $FB6$; thus,

although $FB1$ has an overall higher performance than $FB6$, the latter appears to be more stable, which is due to having more instances per match object, i.e., taking into account more similarity measures.

Table 6.2 presents the results for the other IR evaluation metrics discussed above. These results again confirm the aforementioned observations. For all the considered metrics, $FB5$ and $FB1$ perform better, followed by $FB6$.

7 Conclusions

In this paper we have dealt with the problem of Semantic Web service discovery, focusing on how to exploit user feedback to improve the quality of the search results. This relies on the idea that modern Web 2.0 applications allow users to explicitly express their opinion by giving feedback about available resources, in the form of rating, tagging, etc. We have presented an architecture that allows to collect user feedback on retrieved services and incorporate it in the Semantic Web service matchmaking process. We have proposed different methods to combine this user feedback with the output of matchmaking algorithms in order to improve the quality of the match results. Further, we have discussed how to overcome the problems of a limited amount of feedback or of previously unknown requests (i.e., where no previous feedback is available for the request), by utilizing information from similar requests. We have conducted an experimental evaluation using a publicly available collection of OWL-S services. We have compared our feedback-aware matchmaking strategies to state-of-the-art matchmaking algorithms that do not take feedback into account. Our experimental results show that user feedback is a valuable source of information for improving the matchmaking quality.

Our current and future work focuses mainly on two directions: a) investigating in more detail the social aspects involved in the process of collecting and aggregating user ratings for services, and b) extending our experimental setup with additional scenarios.

Acknowledgements. This work was partially supported by the European Commission in the context of the FP7 projects SYNC3 and GRAPPLE.

References

1. Dong, X., Halevy, A.Y., Madhavan, J., Nemes, E., Zhang, J.: Similarity Search for Web Services. In: VLDB. (2004) 372–383
2. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic Matching of Web Services Capabilities. In: ISWC. (2002) 333–347
3. Klusch, M., Fries, B., Sycara, K.P.: Automated Semantic Web service discovery with OWLS-MX. In: AAMAS. (2006) 915–922
4. Skoutas, D., Sacharidis, D., Simitsis, A., Kantere, V., Sellis, T.: Top-k Dominant Web Services under Multi-criteria Matching. In: EDBT. (2009) 898–909
5. Bao, S., Xue, G.R., Wu, X., Yu, Y., Fei, B., Su, Z.: Optimizing Web Search Using Social Annotations. In: WWW. (2007) 501–510

6. Akkiraju, R., et. al.: Web Service Semantics - WSDL-S. In: W3C Member Submission. (November 2005)
7. Burstein, M., et. al.: OWL-S: Semantic Markup for Web Services. In: W3C Member Submission. (November 2004)
8. H. Lausen, A. Polleres, and D. Roman (eds.): Web Service Modeling Ontology (WSMO). In: W3C Member Submission. (June 2005)
9. Li, L., Horrocks, I.: A Software Framework for Matchmaking based on Semantic Web Technology. In: WWW. (2003) 331–339
10. Cardoso, J.: Discovering Semantic Web Services with and without a Common Ontology Commitment. In: IEEE SCW. (2006) 183–190
11. Skoutas, D., Simitsis, A., Sellis, T.K.: A Ranking Mechanism for Semantic Web Service Discovery. In: IEEE SCW. (2007) 41–48
12. Skoutas, D., Sacharidis, D., Kantere, V., Sellis, T.: Efficient Semantic Web Service Discovery in Centralized and P2P Environments. In: ISWC. (2008) 583–598
13. Bellur, U., Kulkarni, R.: Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching. In: ICWS. (2007) 86–93
14. Hau, J., Lee, W., Darlington, J.: A Semantic Similarity Measure for Semantic Web Services. In: Web Service Semantics Workshop at WWW. (2005)
15. Srinivasan, N., Paolucci, M., Sycara, K.P.: An Efficient Algorithm for OWL-S Based Semantic Search in UDDI. In: SWSWPC. (2004) 96–110
16. Constantinescu, I., Binder, W., Faltings, B.: Flexible and Efficient Matchmaking and Ranking in Service Directories. In: ICWS. (2005) 5–12
17. Colgrave, J., Akkiraju, R., Goodwin, R.: External Matching in UDDI. In: ICWS. (2004) 226
18. Kaufer, F., Klusch, M.: WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. In: ECOWS. (2006) 161–170
19. Balke, W.T., Wagner, M.: Cooperative Discovery for User-Centered Web Service Provisioning. In: ICWS. (2003) 191–197
20. Xu, Z., Martin, P., Powley, W., Zulkernine, F.: Reputation-Enhanced QoS-based Web Services Discovery. In: ICWS. (2007) 249–256
21. Kießling, W., Hafenrichter, B.: Optimizing Preference Queries for Personalized Web Services. In: Communications, Internet, and Information Technology. (2002) 461–466
22. Lamparter, S., Ankolekar, A., Studer, R., Grimm, S.: Preference-based selection of highly configurable web services. In: WWW. (2007) 1013–1022
23. O'Reilly, T.: O'Reilly Network: What is Web 2.0 (September 2005)
24. Abel, F., Henze, N., Krause, D.: Ranking in Folksonomy Systems: Can Context Help? In: CIKM. (2008) 1429–1430
25. Hotho, A., Jäschke, R., Schmitz, C., Stumme, G.: Information Retrieval in Folksonomies: Search and Ranking. In: ESWC. (2006) 411–426
26. Mislove, A., Gummadi, K.P., Druschel, P.: Exploiting Social Networks for Internet Search. In: Workshop on Hot Topics in Networks. (2006)
27. Manning, C.D., Raghavan, P., Schütze, H.: An Introduction to Information Retrieval. Cambridge University Press (2008)
28. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: ISWC. (2002) 54–68
29. Whitby, A., Josang, A., Indulska, J.: Filtering Out Unfair Ratings in Bayesian Reputation Systems. In: AAMAS. (2004)
30. Yu, B., Singh, M.P., Sycara, K.: Developing trust in large-scale peer-to-peer systems. In: IEEE Symposium on Multi-Agent Security and Survivability. (2004) 1–10