Web Service Discovery Based on Past User Experience

Natallia Kokash, Aliaksandr Birukou, Vincenzo D'Andrea

DIT - University of Trento, Via Sommarive, 14, 38050 Trento, Italy
{kokash, birukou, dandrea}@dit.unitn.it

Abstract. Web service technology provides a way for simplifying interoperability among different organizations. A piece of functionality available as a web service can be involved in a new business process. Given the steadily growing number of available web services, it is hard for developers to find services appropriate for their needs. The main research efforts in this area are oriented on developing a mechanism for semantic web service description and matching. In this paper, we present an alternative approach for supporting users in web service discovery. Our system implements the implicit culture approach for recommending web services to developers based on the history of decisions made by other developers with similar needs. We explain the main ideas underlying our approach and report on experimental results.

Keywords. Web Service Discovery, Recommendation Systems, Implicit Culture

1 Introduction

The state-of-the-art in business integration is defined by implementation of the service-oriented vision using web service technology. Web services are loosely coupled, distributed entities that can be described, published, discovered and invoked via the web infrastructure. Three main standards in this area include Web Service Description Language (WSDL) for presenting service interfaces, Universal Description, Discovery and Integration (UDDI) registries for publishing, and Simple Object Access Protocol (SOAP) for message transporting.

With ever increasing number of available web services it is problematic to find a service with required functionality and appropriate quality characteristics. Most of the proposals in the area of web service discovery rely on logically precise semantic descriptions of web services by providers [1][2][3]. Such approaches are efficient only if providers publish exhaustive service specifications. Tools for automatic or semi-automatic semantic annotation can significantly reduce required amount of work, but, in principle, the consumer must trust the provider to deliver the service fully compliant with the description. Additionally, web services or providers can be evaluated by a trusted party, i.e., by a specialized unbiased agency that tests web services, verifies their descriptions (whether there is a discrepancy between specified and implemented features), publishes Quality

of Service (QoS) data, etc. This solution is relatively expensive and inefficient due to its rather static nature. Automated central monitors are complex, and either provide limited monitoring facilities or require involvement of domain-specific logic for verifying web service behavior [4][5].

On the other hand, there are service clients who already have experience in using web services and therefore can help in selecting services with adequate quality. This principle is widely used by (collaborative) recommendation and reputation systems [6][7]. Often web services are oriented not on public use but aim at enabling easy information exchange between a set of partner organizations. Since web services belong to different domains, only a specific set of web services is interesting for a particular consumer. A group of clients with common interests form a virtual *community* where they can exchange the experience, i.e., the knowledge gained after having interaction with a web service. Being a member of such a community can help to reduce the information overload and enhance web service discovery and selection facilities.

In this paper, we present a system for discovery of web services. The system is based on the implicit culture framework [8] and helps developers make a decision about which services to use by getting suggestions from the community. The implicit culture framework has been implemented in the form of a domain-independent meta-recommendation service, the *IC-Service*, that uses web service technology and can be tuned via configuration interface [9]. In our approach, no communication between members of the community is needed, and no explicit ratings of web services are required.

The paper is organized as follows. In Section 2, the basic idea of implicit culture is presented and the configuration of the *IC-Service* for our application is explained. Section 3 provides implementation details, while Section 4 presents experimental results. Related work is analyzed in Section 5, and Section 6 draws conclusions and outlines future work.

2 Implicit Culture

This section presents an overview of the general idea of the implicit culture framework and the System for Implicit Culture Support (SICS). The SICS provides the basis for the *IC-Service* we have used to provide recommendation facilities.

The behavior of a person in an unknown environment is far from optimal. There exist many situations where it is difficult to take the right decision due to the lack of knowledge. This might not be the case for experienced people who have previously encountered similar problems and have acquired the necessary knowledge. The knowledge about acting effectively in the environment is often implicit (i.e., highly personalized) and specific to the community. Therefore, this knowledge could be referred to as a *community culture*. The idea behind the implicit culture framework is that it is possible to elicit the community culture by observing the interactions of people with the environment and to encourage the newcomer(s) to behave in a similar way. Implicit culture assumes that agents perform actions on objects, and the actions are taken in the context of situations,

so agents perform situated actions. The "culture" contains information about actions and their relation to situations, namely which actions are usually taken by the observed group and in which situations. This information is then used to produce recommendations for other agents. When newcomers start to behave similarly to the community culture, it means that a *knowledge transfer* occurred. The goal of the SICS is to perform such transfer of knowledge.

The basic architecture for the SICS [8] consists of the following three components: the *observer* module, which records the actions performed by the client during the use of the system; the *inductive* module, which analyzes the stored observations and implements data mining techniques to discover behavior patterns; the *composer* module, which exploits the information collected by the observer module and analyzed by the inductive module in order to produce recommendations.

In terms of our problem domain, the observer saves the following information: the user request (a textual description and characteristics of a required web service), the context in which the request occurred, the services proposed as a solution, the service chosen and invoked by the user, and, finally, the result of the invocation (successful web service invocation, exception raised, etc.). Then, the request-solution pairs that indicate which web services are selected for which requests could be determined by analyzing the interaction history between users and the system. This step is performed by the inductive module and it is now omitted. Finally, the composer module matches the user request with web services by calculating the similarity between the request given by the user and the requests that users provided previously, and by selecting web services chosen for the most similar past requests.

The described schema is implemented within the *IC-Service* [9] that provides recommendation facilities based on implicit culture. The configuration of the *IC-Service* for our application is shown in Figure 2. Along with the *SICS core*, which forms the main part of the service, the *IC-Service* includes the other two important components: the *remote client* and the *remote module*. The recommendation system is available as a web service that can be accessed via the remote client. The remote client presents a wrapper that hides protocols used for information exchange with the SICS. The remote module defines protocols for information exchange with the remote client from the direction of the SICS and converts the objects of the SICS core in the format compatible with these protocols. We refer the reader interested in the details of the modules to the description of the *IC-Service* [9].

In the current version of our system we do not use the inductive module to infer new behavior patterns, but predefine them manually. The *IC-Service* allows for the adjustment of a recommendation strategy through configuring theory rules. A theory rule is defined as follows:

$$\text{if } \textit{consequent}(\text{predicates}) \text{ then } \textit{antecedent}(\text{predicates}),$$

where predicates describe either conditions on observations (action-predicates) or conditions on time (temporal-predicates). Each predicate may include several

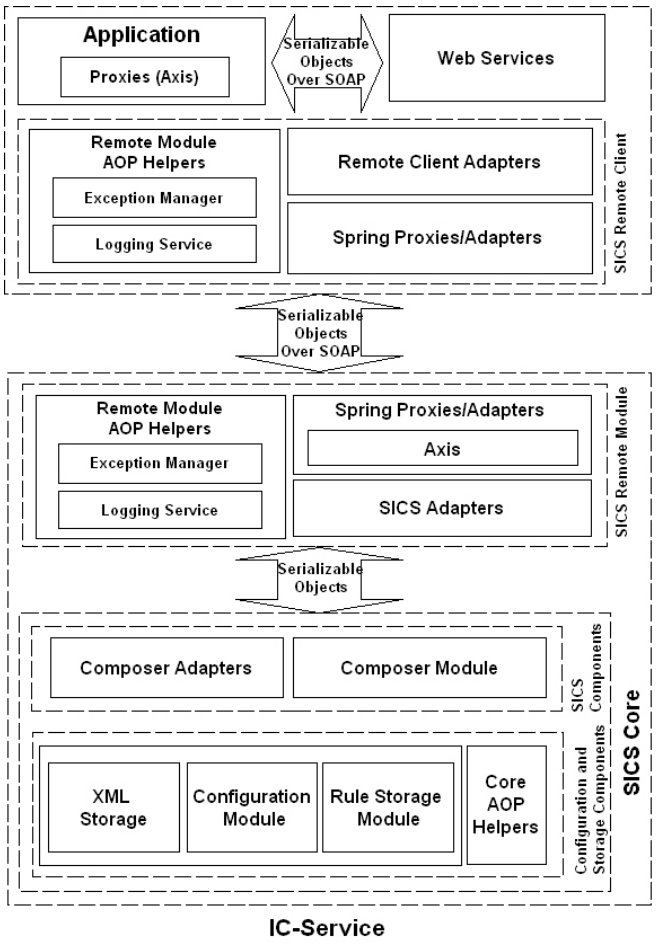


Fig. 1. Configuration of the *IC-Service* for web service discovery

action-rules, which specify patterns on actions, agents, objects, scenes[1], and their attributes. Observations from the SICS storage are analyzed by the composer module according to these patterns. For matching of the discovered observations a similarity algorithm must be defined. The *IC-Service* provides a simple matching method that compares pairs of observations using predefined similarity values for their elements (actions, objects, etc.). These values can be configured for each particular type or for each particular instance of the element. A similarity threshold for matching also can be tuned. In addition, a plug-in mechanism enables the possibility of involving other similarity algorithms.

[1] a *scene* is the architectural abstraction of a situation

Table 1. Actions observed by the system.

Action	Agents	Objects
invoke	application	operation, input
get_response	application	operation, output
raise_exception	application	operation, input
provide_feedback	application, developer	operation, rate
submit_request	application, developer	request

3 A System for Web Service Discovery

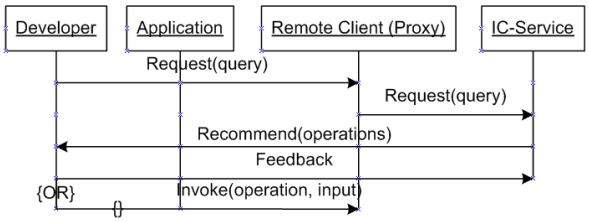
This section gives a description of the process of web service discovery within the system.

The motivation for adopting the implicit culture approach for web service discovery stems from the difficulty of developers in finding and selecting web services suitable for their applications [10]. The system is intended for the use by a virtual community, giving suggestions about web services suitable for this community. In our domain, developers and their applications perform actions on web services. Types of actions analyzed by the SICS are presented in Table 1 and will be explained later in more detail. Actions, agents and objects also may have multiple attributes, i.e., features helpful for their analysis. For example, information about a web service (id, name, provider, etc.) is stored as an attribute of an object *operation*. The description of the complete set of the stored attributes is not needed for understanding the current paper and therefore is omitted here.

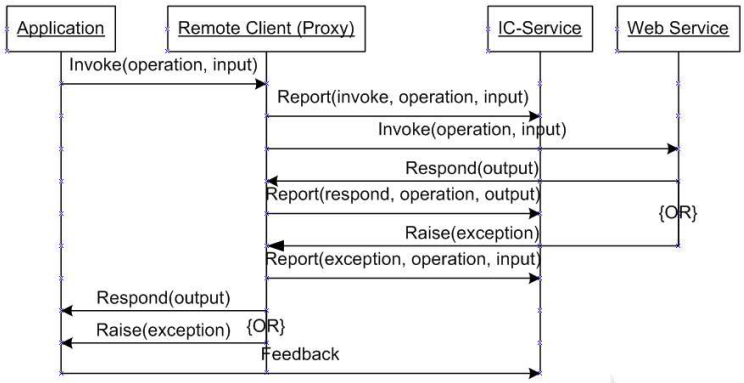
In order to use the system, each user must install a remote client. The goal of the remote client is to communicate with the SICS, in particular, forward user requests and store observations about user actions, applications and behavior of web services. To enable observations of interaction with a web service we have extended the `JavaStubsWriter` class of the open-source Apache Axis framework[2]. This class generates stubs for web service invocation. The modification that we implemented allows the stubs to report the information about the communication between a user application and a web service to the *IC-Service*, using the remote client. Thus, to join a virtual community that shares experience in retrieving of web services, the user must (1) install the remote client, (2) generate stubs for service invocations using the modified version of the Axis tool. No further intervention, user-to-user or user-to-system communications are required, except for submitting requests. If the user does not need to search for new web services, the system can be used for service monitoring on the client side. Run-time web service monitoring is essential for real-world service-oriented systems where control of service quality is needed [11][12].

The SICS remote client provides an interface for the user to access the system by submitting requests. Request may include textual description of the goal, name of the desired operation, description of its input/output parameters, description of a desired web service and its features (provider, etc.). By configuring the similarity algorithm it is possible to define whether these requirements are

[2] http://xml.apache.org/axis/



(a) Search process



(b) Monitoring process

Fig. 2. Sequence diagrams.

considered as strict (only services that meet them are recommended) or as preferred (services that better fit the request than others are recommended).

The search scenario is given in Figure 2(a). A user submits a request via the remote client, from where the request is forwarded to the *IC-Service*, and a list of recommended services is returned. The feedback is collected via the optional *provide_feedback* action, which expresses the level of user satisfaction with the result, or through the *invoke* action, which marks a service as suitable for the request. If the user decides to use one of the services, the further information is acquired. The *get_response* action marks a service as available and the *raise_exception* action signals that the service is not available or faulty. The monitoring process is shown in Figure 2(b). In short, when the application invokes some operation provided by a web service, the remote client reports to the *IC-Service* on the *invoke* action. Similarly, when the web service sends a response message or raises an exception, the remote client reports to the *IC-Service* on the *get_response* or *raise_exception* action, respectively. Having received a response message, the user application can generate a feedback based on an extra-knowledge about the expected result (e.g., the feedback is positive if the meaningful output has been obtained, etc.)

The *IC-Service* processes the query from the system within two steps. In the first step, the action contained in the query, i.e. the *submit_request* action, is matched with the theory to determine the next action that must follow, i.e. the *invoke* action. In the second step, the SICS finds situations where the invoke action has been previously performed, determining web service operations used for similar requests in the past. In this step, the similarity between the current user request and the previously submitted requests is calculated. As a result, the *IC-Service* returns a set of services that have been used for similar requests in the past.

Let us illustrate how the search process takes place in our example. The user submits the request represented by the following query:

Goal : Get weather forecast for Rome (this is in Italy);
Operation : Get weather;
Input : City name, country name;
Output : Weather forecast (temperature, humidity, etc.).

The *IC-Service* matches the request action with the theory, which contains rules of the following form:

if *submit_request*(request) then *invoke*(operation-X(service-Y), request).

This means that the *invoke* action must follow the *submit_request* action and both actions are related to the same query. The SICS matches the request action with the request part of the theory, and searches for situations where the invoke action has been performed. It finds the following situations:

ID	Action	Goal	Operation
1	invoke	get weather report for all major cities around the world	getWeather (service = GlobalWeather)
2	invoke	get conversion rate from one currency to another currency	conversionRate (service = CurrencyConvertor)
3	invoke	return the weather for a given US postal code	getWeatherByZip (service = DOTSFastWeather)

As a result, the SICS recommends that the user invokes either getWeather operation of the GlobalWeather web service or getWeatherByZip operation of the DOTSFastWeather web service. Having analyzed the proposed results, the user invokes the former operation. After observing the *invoke* action, this service will be marked as suitable for the above query. Further, it may be considered relevant for requests asking for information about Italy.

Note, that instead of the *invoke* action, the *get_response* action can be put in the theory. In this case, only web services invoked successfully at least once will be considered. The same mechanism can be used for reputation-based web service filtering: users can explicitly rate services using *provide_feedback* action.

The *IC-Service* enables saving various information and defining inferring rules and similarity measures on them. In the context of the presented system, the *IC-Service* is used to collect reports of service invocations by clients, to keep previous user requests, and to define similarities between users based on the information about the services they use. The implemented schema can be extended to store other important information about web services, such as cases

of Service Level Agreement (SLA) [13] violation or measurements of QoS parameters. This information further can be dynamically involved in the refined web service discovery and selection through defining new theory rules. The meta-data about web services, augmented with the help of our system, can be further used for hybrid web service matching algorithms [14]. In the simplest scenario, the remote client can submit user requests to the UDDI registry through `UDDI4J`[3] API in order to get information about recently appeared web services.

4 Experimental Evaluation

The goal of the experiment is to evaluate the performance of the system. We have defined user profiles in order to simulate the behavior of real users. A user profile contains a set of queries and a set of web service operations relevant to these queries. The set of queries is exploited to simulate the request-generation behavior by choosing and submitting a query randomly, while the set of web service operations is used to simulate the result-selection behavior by selecting one of the operations. Each query consists of a brief natural language description of the desired operation. The intuition behind the user profile is as follows: the user submits a request for a service operation. After getting suggestions, (s)he will invoke one of the operations (s)he considers relevant. This invocation is monitored by the remote client of the *IC-Service*. The choice of the user that submits a request to the system in a given moment is random.

The quality of recommendations is measured using the precision, recall and F-measure [15]:

$$\text{Precision} = \frac{\text{Relevant} \cap \text{Retrieved}}{\text{Retrieved}} \quad \text{Recall} = \frac{\text{Relevant} \cap \text{Retrieved}}{\text{Relevant}} \quad \text{F} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The precision measures the fraction of relevant items among those recommended. The recall measures the fraction of the relevant items included in the recommendations. The F-measure is a tradeoff between these two metrics.

Since internally suggestions are filtered by the composer module of the SICS within the *IC-Service*, the precision in our case depends on the similarity measure adopted in the composer module. The recall in our settings demonstrates how the system learns from past experience.

In the experiment we used the Vector Space Model with Term Frequency - Inverse Document Frequency (TF-IDF) metric in the composer module to calculate the similarity between queries. More precisely, a query in this model is represented as a sequence of terms, $q = (t_1, t_2, ..., t_{|q|})$, where $|q|$ is the length of the query and $t_j \in T, j = \overline{1, |q|}$. T is a vocabulary of terms, containing all terms from the collection of queries $Q = \{q_1, ..., q_n\}$ submitted to the system, where n is a total number of queries. For each term t_j let n_{ij} denote the number of occurrences of t_j in q_i, and n_j the number of queries that contain t_j at least once.

[3] http://uddi4j.sourceforge.net/doc.html

Table 2. Experimental collection

Category	Web service	Operation
Currency	{http://www.webserviceX.NET/}CurrencyConvertor	conversionRate
	{http://www.xmethods.net/sd/}CurrencyExchangeService	getRate
	{http://www.myasptools.com/}currencyWS	getRate
	{http://www.xignite.com/services/}XigniteCurrencies	getLatestCrossRate
DNA	{http://www.themindelectric.com/wsdl/Blast/}Blast	searchSimple
	{http://www.themindelectric.com/wsdl/Fasta/}Fasta	searchSimple
	{http://www.themindelectric.com/wsdl/TxSearch/}TxSearch	searchSimple
	{http://www.themindelectric.com/wsdl/SRS/}SRS	searchSimple
SMS	{http://www.webserviceX.NET}SendSMSWorld	sendSMS
	{http://www.sms.mio.it/webservices/sendmessages.asmx}	sendSMS
	{http://ws.AcrossCommunications.com/}SMS	SendEx
	{http://SMSServer.dotnetISP.com}ServiceSMS	sendSmsText
Weather	{http://www.webserviceX.NET}GlobalWeather	getWeather
	{http://ejse.com/WeatherService/}Service	getWeatherInfo
	{http://www.myasptools.com/}WeatherFetcher	getWeather
	{http://www.serviceobjects.com/}DOTSFastWeather	getWeatherByZip
ZIP	{http://www.jasongaylord.com/webservices/zipcodes}ZipCodes	zipCodesFromCityState
	{http://ripedev.com/xsd/ZipCodeResults.xsd}ZipCode	cityToZipCode
	{http://webservices.eraserver.net/}ZipCodeResolver	shortZipCode
	{http://www.webserviceX.NET}USZip	getInfoByCity

For calculating the TF-IDF weight of the term t_j in the query q_i and defining the similarity between queries q_i and q_k the following formulas are used:

$$w_{ij} = \frac{n_{ij}}{|q_i|} * \log(\frac{n}{n_j}), \quad cos(w_i, w_k) = \frac{{w_i}^T w_k}{\sqrt{{w_i}^T w_i}\sqrt{{w_k}^T w_k}}.$$

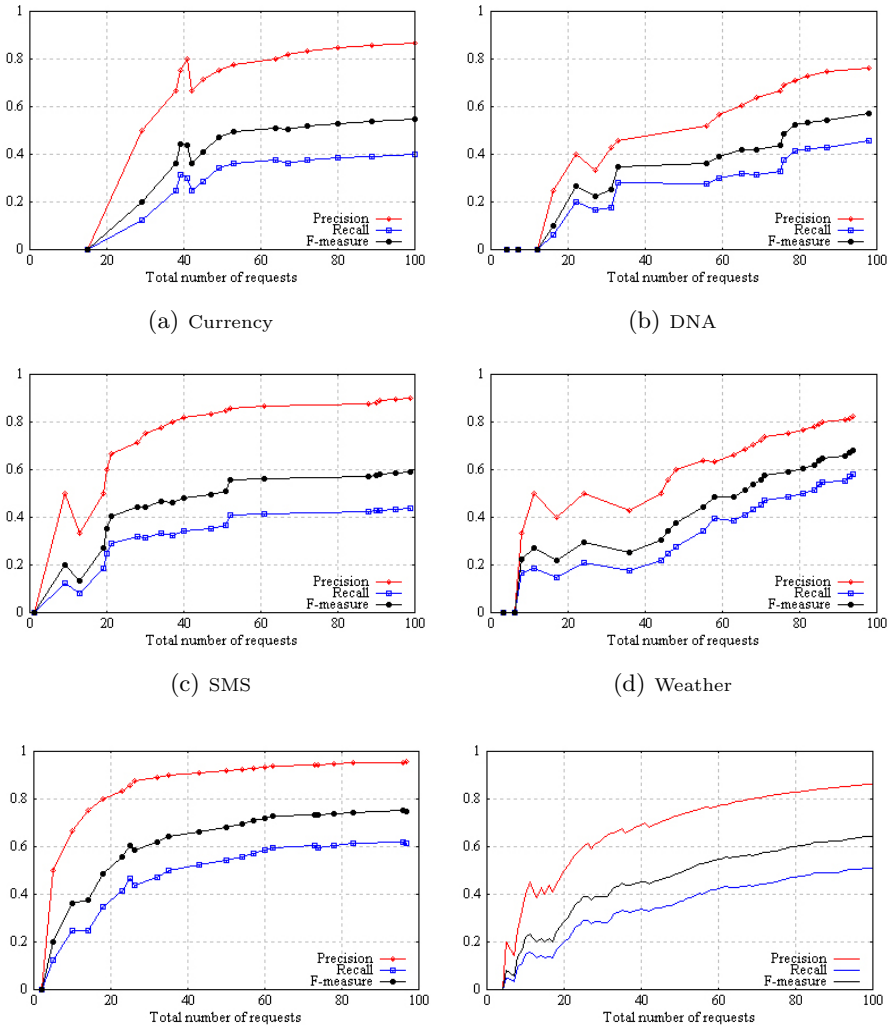
Here $w_i = (w_{i1}, ..., w_{im}), w_k = (w_{k1}, ..., w_{km})$ denote vectors of TF-IDF weights corresponding to the queries q_i and q_k, and m is the length of the vocabulary.

In the experiment we used a collection of 20 web services from XMethods.com, divided into five topic categories (see Table 2). For each category we chose four semantically equivalent operations and formed 20 queries based on their short natural language descriptions from WSDL files. The number of the users in the experiment is equal to four and the number of requests submitted to the system is equal to 100.

The results of the simulations are given in Figure 4. The precision, recall, and F-measure of the recommendations of the SICS for each of the five groups are given and the average performance of the system for all requests is drawn. According to these results, the precision, recall and F-measure of the system tend to increase with the number of user requests. This is justified by the fact that the number of observations about past selections in the system also increases and, as a result, the SICS has more information for the analysis. We can see that just after 20 searches the precision reaches and maintains a quite high level.

5 Related Work

The idea of applying collaborative filtering to web service selection appeared in the literature several times, see papers by Kerrigan [16] and Sherchan [17] for example. Most of the approaches consider ratings of service providers based on



(a) Currency (b) DNA

(c) SMS (d) Weather

(e) ZIP (f) Total

Fig. 3. Performance of the system For five groups of similar web services, the precision, recall and F-measure are calculated for the first 100 requests submitted to the system by four clients.

subjective opinions of web service users. Manikrao and Prabhakar [18] describe a web service selection framework which combines a recommendation system with semantic matching of service requirements. The approach is based on user feedback and collaborative filtering techniques and is oriented towards helping a user to select a web service from a set of similar services. When the user invokes a web service, the system asks the user to rate the service. However, the previous research has shown that users are very unlikely to provide explicit ratings [19].

Alternatively, user profiles can be obtained by implicitly observing user interactions with the system. Maximilien et al. [20] propose an agent-based framework where agents act as proxies to collect information and to build the reputation of semantic web services. Agents are used to manage available service resources: an agent acting on behalf of the owner looks for services and evaluates possible choices. Three-level ontology is proposed to model quality issues of services. Another multi-agent framework for QoS-based web service selection is proposed by Wang et al. [21]. The authors present a distributed reputation assessment algorithm for QoS support.

The problem of unfair ratings is typical for such kind of systems. However, there exist also approaches that can successfully eliminate ratings from malicious agents [22]. The underlying idea in this approach is to associate ratings with some level of quality and ignore the ratings with associated quality below a certain threshold and ratings from the clients that have a credibility below a certain threshold. For example, credibility of a newcomer may grow up with number of the reports compliant with the reports from other clients. Sherchan et al. [17] analyze user rating behavior to infer the rationale for ratings in a web services environment.

Casati et al. [23] present a system for dynamic web service selection based on data mining techniques. The authors analyze past executions of the composite web services and build a set of context-sensitive selection models to be applied at each stage in the composite service execution.

The idea of using monitored data and/or past user experience for collaborative QoS-driven web service selection is examined in several research works. In our system, we use such an approach to match web services with user requests. The fact that the recommended web services are exploited by other users guarantees a certain level of their quality. The idea behind this is that web services with low quality do not have many clients. Our system allows for the reuse of experience by new service consumers through considering past behavior of users in similar situations. In this way, non-existent, often unavailable, incomplete or faulty services (whose descriptions, however, may be published in the registries) are filtered.

6 Conclusions and Future Work

We have presented a recommendation system that facilitates the discovery of web services satisfying user needs. The system is based on the implicit culture framework that uses the history of user-system interactions and client-service

communication logs to provide recommendations on web services. It can be used to enhance the retrieval API of service registries.

Future work includes the implementation and evaluation of more complex recommendation scenarios such as collaborative service testing through mining dependencies between exceptions of a specified type and sort of input data, web service monitoring and QoS-based selection. We are planning to run experiments with a semantic method for matching observations implemented in the system. In perspective, the ability of inductive module to infer behavior patterns for web service discovery will be evaluated.

Acknowledgements

We are grateful to Prof. Enrico Blanzieri for his interest in our work and many fruitful discussions we had together.

References

1. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K.: Web service semantics - WSDL-S, available at http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.pdf (2005)
2. Keller, U., Lara, R., Polleres, A.: WSMO web service discovery. WSMO working draft, available at http://www.wsmo.org/2004/d5/d5.1/. (2004)
3. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., University, Y., et al.: OWL-S: Semantic markup for web services. W3C member submission, available at http://www.w3.org/Submission/OWL-S/ (2004)
4. Piccinelli, G., Stefanelli, C., Trastour, D.: Trusted mediation for e-service provision in electronic marketplaces. In: Proceedings of the International Workshop on Electronic Commerce. Volume 2232 of LNCS., Springer (2001) 39 – 50
5. Mahbub, K., Spanoudakis, G.: A framework for requirements monitoring of service based systems. In: Proceedings of the International Conference on Service-Oriented Computing (ICSOC), ACM Press (2004) 84 – 93
6. Herlocker, J.L., Konstan, J.A., Riedl, J.: Explaining collaborative filtering recommendations. In: Proceedings of ACM Conference on Computer Supported Cooperative Work, ACM Press (2000) 241–250
7. Maximilien, E.M., Singh, M.P.: Conceptual model of web service reputation. SIGMOD Record **31**(4) (2002) 36–41
8. Blanzieri, E., Giorgini, P., Massa, P., Recla, S.: Implicit culture for multi-agent interaction support. In: Proceedings of the International Conference on Cooperative Information Systems (CooplS). Volume 2172 of LNCS., Springer (2001) 27–39
9. Birukou, A., Blanzieri, E., D'Andrea, V., Giorgini, P., Kokash, N., Modena, A.: *IC-Service*: A service-oriented approach to the development of recommendation systems. In: Proceedings of ACM Symposium on Applied Computing. Special Track on Web Technologies, ACM Press (2007)
10. Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A.: Web service discovery mechanisms: Looking for a needle in a haystack? In: International Workshop on Web Engineering. (2004)

11. Tian, M., Gramm, A., Ritter, H., Schiller, J.: Efficient selection and monitoring of QoS-aware web services with the WS-QoS framework. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, IEEE Computer Society (2004) 152 – 158
12. Bostrom, G., Giambiagi, P., Olsson, T.: Quality of service evaluation in virtual organizations using SLAs. In: International Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems (IS-TSPQ). (2006)
13. Dan, A., Davis, D., Kearney, R., et al.: Web services on demand: WSLA-driven automated management. IBM Systems Journal **43**(1) (2004) 136–158
14. Kokash, N., van den Heuvel, W.J., D'Andrea, V.: Leveraging web services discovery with customizable hybrid matching. In: Service-Oriented Computing - ICSOC 2006. Volume 4294 of LNCS., Springer (2006) 522–528
15. Baldi, P., Frasconi, P., Smyth, P.: Modeling the Internet and the Web: Probabilistic Methods and Algorithms. Wiley (2003)
16. Kerrigan, M.: Web service selection mechanisms in the web service execution environment (WSMX). In: Proceedings of the ACM Symposium on Applied Computing (SAC), ACM Press (2006) 1664–1668
17. Sherchan, W., Loke, S.W., Krishnaswamy, S.: A fuzzy model for reasoning about reputation in web services. In: Proceedings of ACM Symposium on Applied Computing, ACM Press (2006) 1886 – 1892
18. Manikrao, U.S., Prabhakar, T.: Dynamic selection of web services with recommendation system. In: Proceedings of the International Conference on Next Generation Web Services Practices (NWESP), IEEE Computer Society (2005) 117
19. Claypool, M., Le, P., Wased, M., Brown, D.: Implicit interest indicators. In: International Conference on Intelligent User Interfaces, ACM Press (2001) 33–40
20. Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic web services selection. IEEE Internet Computing **8**(5) (2004) 84–93
21. Wang, H., Yang, D., Zhao, Y., Gao, Y.: Multiagent system for reputation–based web services selection. In: International Conference on Quality Software (QSIC), IEEE Computer Society (2006) 429–434
22. Xu, P., Gao, J., Guo, H.: Rating reputation: A necessary consideration in reputation mechanism. In: Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, IEEE Computer Society (2005)
23. Casati, F., Castellanos, M., Dayal, U., Shan, M.C.: Probabilistic, context-sensitive, and goal-oriented service selection. In: Proceedings of the Internationa Conference on Service-Oriented Computing (ICSOC), ACM Press (2004) 316–321