A Flexible Tool for Web Service Selection in Service Oriented Architecture

A Flexible Tool for Web Service Selection

Walaa Nagy, Hoda M. O. Mokhtar, Ali El-Bastawissy
Faculty of Computers and Information
Cairo University
Cairo, Egypt

***Abstract*--Web Services are emerging technologies that enable application to application communication and reuse of services over Web. Semantic Web improves the quality of existing tasks, including Web services discovery, invocation, composition, monitoring, and recovery through describing Web services capabilities and content in a computer interpretable language. To provide most of the requested Web services, a Web service matchmaker is usually required. Web service matchmaking is the process of finding an appropriate provider for a requester through a middle agent. To provide the right service for the right user request, Quality of service (QoS)-based Web service selection is widely used. Employing QoS in Web service selection helps to satisfy user requirements through discovering the best service(s) in terms of the required QoS. Inspired by the mode of the Internet Web search engine, like Yahoo, Google, in this paper we provide a QoS-based service selection algorithm that is able to identify the best candidate semantic Web service(s) given the description of the requested service(s) and QoS criteria of user requirements. In addition, our proposed approach proposes a ranking method for those services. We also show how we employ data warehousing techniques to model the service selection problem.**

The proposed algorithm integrates traditional match making mechanism with data warehousing techniques. This integration of methodologies enables us to employ the historical preference of the user to provide better selection in future searches. The main result of the paper is a generic framework that is implemented to demonstrate the feasibility of the proposed algorithm for QoS-based Web application. Our presented experimental results show that the algorithm indeed performs well and increases the system reliability.

Keywords-Semantic Web; Web services; Web services match-making; Data warehouses; Quality of Services (QoS); Web service ranking.

I. INTRODUCTION

A. Background

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web [1]. With the

development of service-oriented architecture (SOA) and cloud computing, more and more services are continuously emerging on the Internet, such as Amazon EC2[1], Google App Engine[2]; thus, it is expected that in the near future there would be more and more different types of services, and more and more number of services emerging on the Internet.

Besides as the users often do not know how to quantify the trade-offs between different Web services and just wish to quickly grasp what can be potentially interesting, a single solution that is the best one from an objective point of view typically does not exist; instead, many reasonable alternative services usually exist .

Hence, as both the user requirements, and the number of available services and service providers increases, improving the effectiveness and accuracy of Web service discovery and selection mechanisms becomes a crucial issue [2, 3].

Today, the Universal Description, Discovery and Integration UDDI standard is considered the most commonly used service discovery standard [4]. However, UDDI has 2 main shortcomings: first, it returns coarse results for a keyword based search, and second, more importantly it lacks semantics. Hence, UDDI is basically a framework that supports category based search [4].

On the other hand, semantic Web improves the quality of existing tasks, including Web services discovery, invocation, composition, monitoring, and recovery by describing Web services capabilities and content in a computer interpretable language [4].

One of the main applications of semantic Web is its usage in the semantic Web services in the matchmaking process. Matchmaking is the process of finding an appropriate provider for a requester through a middle agent. Consequently, Semantic matchmaking is used by semantic Web services to find valuable service candidates and selecting the most suitable service(s) that best match user request [5, 6]. In this work we use OWL-S [7] for describing the used services.

OWL-S is an OWL-based Web service ontology, which supplies a core set of markup language, constructs for describing the properties and capabilities of Web services in

[1]http://aws.amazon.com/

[2]http :// code.google.com/appengine/

an unambiguous, computer-interpretable form. The overall ontology consists of three main components: *the service profile* for advertising and discovering services; *the process model*, which gives a detailed description of a service's operation; *and the grounding*, which provides details on how to interoperate with a service, via messages. Specifically, it specifies the signature that is composed of the inputs required by the service, and the outputs generated. Furthermore, since a service may require external conditions to be satisfied, and its execution can change those conditions, the profile describes the preconditions required by the service and the expected effects that result from the execution of the service. For more details, we refer the reader to for example [7].

Nevertheless, with the increasing number of Web services providing similar functionalities, the QoS (Quality of Service) is becoming an important criterion of selection of the best available service. Although, we believe that designing intuitive, easy-to-use user interfaces can help the process of collecting user feedback and preferences; in this work, we do not deal with this issue, instead our focus is on how the collected information in the user profile is processed and integrated in the selection process of Web services to improve the results of subsequent searches.

Inspired by the fact that current enterprise decision making systems benefit more from OLAP, and data warehouse techniques [8], in this paper we show how we can adopt the power of data warehousing in supporting decision making, and how data warehouses and OLAP techniques can help in selecting the most interesting result with the above issues being considered.

In general, data warehousing is one of the most common business intelligence tools nowadays. Data warehouses provide a solid platform that includes both current and historical data [8]. Using this platform, companies can therefore make a series of analysis that can help in providing the right service that matches the user request more easily, accurately, and efficiency.

B. Motivation

The availability of service providers with different features makes the task of selecting an appropriate service provider for a user more and more complex which motivates us to consider new solutions for the Web services selection problem in SOA systems. As shown in Table I there are various types and number of services, associated with different performances, prices, platform/APIs, and availability levels [9, 10, 11].

Analyzing these services we find that:

1) There exist various types of services (for example compute, storage etc).

2) There exist a large number of functionally similar services which results in a proliferation in the number of services that provide similar functionality with different QoS criteria (i.e. price, availability).

3) There are a large number of service providers that are continuously emerging on the internet such as Google AppEngine.

4) Finally, there is a wide range in service performance and price. Where different providers offer their services with different prices and performance values.

From that we can conclude that Web service selection process needs five crucial issues:

a) ***Accuracy****: the algorithm should avoid the loss of Web services that can match the user request but their interface is not the same as the user request. Thus, semantic matchmaking of Web services is needed.*

b) ***Flexibility****: new evolving mechanisms should be flexible to support large numbers of services providers.*

c) ***Scalability****: selection algorithm of Web services should be scalable to support any number of QoS requirements.*

d) ***Generality****: the selection algorithm should be as generic as possible to support different users and various user requirements, rather than specific types of users.*

e) ***User personalization****: the algorithm should be able to provide the right service to the right user request; ideally the user preferences should be captured automatically*

C. Our Contribution

Inspired by importance of the Web service selection problem and its vital role in satisfying the requests of billions internet users, in this paper, we address the service selection problem. We focus on the five challenges that we presented earlier namely, the accuracy, flexibility, scalability, generality, and personalization. Our main contributions are as follows:

1) We propose a new service selection algorithm that uses a semantic matchmaker to enhance the selection accuracy.

2) We include QoS criteria in our selection process to find the service that best matches the user requirements and constraints.

3) We employ data warehousing techniques to capture the historical user profile to provide a better service personalization based on previous user requirements and selections.

4) We experimentally show that the proposed algorithm enhances the quality and efficiency of the selection process.

The paper is organized as follows: Section II presents an overview of previous work. Section III describes QoS properties that will be used. Section IV discusses the system architecture. Our proposed selection methodology of Web services is presented in section V. Experimental evaluation is provided in section VI. Finally, section VII concludes our work and presents directions for future work.

Service Provider	Service Type	Price	Platform/API	Availability(SLA)%
Google AppEngine	Compute	$8/application	Java/Spring/Python	99.9
Azure compute (small)	Compute	0.12/h	Windows server2008	99.95
IBM cloud (Unres. Bronze)	Compute	$0.210/ h	RedHat Linux	99.5
AWS SimpleDB	Database	$0.250/GB/Month		
Azure storage	Storage	$0.15 /GB/Month		99.9

TABLE I. SERVICES PROVIDED BY REPRESENTATIVE PROVIDER

II. RELATED WORK

Today we are witnessing a proliferation in the number of available Web services; this proliferation increases the need for automatic Web service retrieval algorithms. Currently, Web service discovery is a challenging task specially when finding the services that match users' interest. This challenge is a natural consequence of the inability of service discovery processes to resolve ambiguities introduced by Web service interfaces. Unfortunately, many of the existing discovery models restrict themselves to finding Web services solely based on the descriptions available within WSDL documents [12, 13].

Several approaches have been proposed in the literature for discovering Web services. In [14], the authors proposed a system that discovers Web services based on keyword matching by taking advantage of the IR technique utilizing Vector Space Model (VSM). This approach computes the similarity between query terms and the document collection focusing mainly on WSDL operations (e.g. operation names).

In a similar effort, the authors in [15] proposed Woogle, a search engine which focuses on retrieving WSDL operations retrieving WSDL operations. Woogle (which discontinued its services in 2006), collected services from accessible service registries and provided clients with capabilities to perform keyword-based search. However, the main underlying concept behind the method implemented in Woogle was based on the assumption that Web services belong to the same domain of interest and are equal in terms of their behavior in accomplishing the required functionality.

In [16], the author provided a comprehensive list of QoS parameters that cover the quality in Web services, and classified them into categories including: (1) runtime- QoS attributes, such as scalability, capacity, performance, reliability, availability, robustness, accuracy, and exception handling; (2) transactional- QoS which mainly focuses on the quality of transactions executed (integrity); (3) configuration management and cost-QoS properties that are to standards and cost, such as regulatory, supported standards, stability, cost, and completeness; and (4) security- QoS properties that are to security, such as authentication, confidentiality, accountability, data encryption, traceability, and non-repudiation.

Other researchers have provided similar lists of QoS properties [17, 18, 19], however, little or no details are given on how to calculate or compute the proposed QoS parameters. Recently, a number of approaches were proposed that presented experimental frameworks that attempt to provide QoS measurements and support for Web services. One of the most common frameworks is QoS Certifier introduced in Ran [16] in which a system is proposed for adding QoS information in UDDI registries using a QoS certification framework. The QoS Certifier verifies QoS claims provided by a service provider. Although the proposed solution may provide QoS support for Web service discovery, it has several limitations such as the redundancy of performing QoS measurements which first have to be supplied by the service provider at the time of registration, and then those QoS measurements will eventually be performed by a certification authority. In addition, this solution proposed a major change to the UDDI specification [20] which is problematic at this stage.

In [21], the authors used the concept of classes in their proposed approach named WS-QoS. WS-QoS attempts to address issues, such as service selection and monitoring of QoS for Web services. WS-QoS not only defines several QoS parameters, but also includes network-level QoS parameters such as packet-loss, and network delay [21]. However, in real world, it is likely that clients would be more interested to know the overall QoS of a Web service, and not network-level details.

In [22], the authors proposed QoS support for service-oriented middleware (SOM). In this model, the middle-ware monitors QoS metrics for Web services automatically, and four QoS properties were identified: time, cost, reliability, and fidelity. Similarly, in [23], the authors proposed a model for identifying services based on QoS guarantees. However, in both of these proposed solutions, the authors did not provide an actual implementation of the proposed systems or how QoS metrics are conducted.

Other approaches focused on the semantic support for Web services as presented in [24], the authors proposed a novel approach to integrate services considering only their availability, the functionalities they provide, and their non-functional QoS properties rather than considering the users direct requests. In [25], the authors proposed a solution for this problem and introduced the Web Service Relevancy Function

(WSRF) that is used for measuring the relevancy ranking of a particular Web service based on QoS metrics and client preferences. However one of the challenges in this work is the clients ability to control the discovery process across accessible service registries for finding services of interest, yet semantic matching of services has not been considered.

In [26] the authors proposed heuristic algorithms that can be used to find a near-to-optimal solution more efficiently than exact solutions. The authors proposed two models for the QoS-based service composition problem. Despite the significant improvement of these algorithms compared to exact solutions, both algorithms do not scale with respect to an increasing number of Web services and remain out of the real time requirements.

Unfortunately many of the existing solutions do not provide ways for clients to articulate service queries tailored to their needs. In fact, the existing discovery models do not sufficiently consider end-to-end discovery mechanisms that can provide clients with quality Web services. In addition, existing QoS discovery models do not provide ways to conduct QoS measurements in a transparent and fair manner. in addition, users do not know the history of the service as if it is a reliable service or not. As a result, the user has to try to use the service and see if it can actually provide the required information or not.

Inspired by the mode of the Internet Web search engine, like Yahoo, Google, the authors on [9] design the service providers search engine (SPSE) algorithm. Different with the existing works, which directly schedule the jobs to resources, the algorithm does not make any schedule decision for the job, but is an assistant tool for service selection.

Our algorithm most similar to his idea but in our algorithm after we select the available providers that can match the user request we enhance the result by conducting semantic matching of services which provide more accurate results also we provide a new method for the selection and ranking of the results of our algorithm. In this work we present a solution that aims to overcome many of the limitations of the existing solutions and offers a novel quality-driven discovery and ranking of Web services. Unlike many of the existing QoS discovery models which require major changes to be made to existing standards such as UDDI, our model serves as an assistant tool for service selection.

Our proposed model measures service qualities in an independent and transparent manner, and allows clients to control and manage the discovery process based on QoS properties.

III. QUALITY OF SERVICE (QOS)

QoS criteria are used to differentiate the Web services providing the same functionality during the service selection process. As a user request can be answered by multiple functionally similar services with different level of QoS. One or more non-functional properties can be associated to a Web service. In this work we use the generic QoS criteria as the basis for further discussions which are also used in [9, 27, 28]. In the rest of this section we explore those additional QoS measures in more details.

- **Trust degree:** Trust degree is a kind of a social attribute of the service provider, and implies its reliability or availability level. The authors in [9] proposed to compute the trust degree of each service provider by aggregating several factors, such as, success rate, user´s evaluation to the service, availability level in Service Level Agreement (SLA). Calibrating them into a decimal value between 0 and 1, and denoting by *TDi* the ith correlative factor. the trust degree of a service provider can be calculated as:

$$\boldsymbol{Trust} = \sum_{\mathbf{i=1}}^{\mathbf{d}} \mathbf{w_i} \times \mathbf{TD_i} \quad (1)$$

Where w_i is the weight for the corresponding factor, and d is the number of factors. We assume that trust degree is used as a decimal score, with a greater value representing more reliable provider.

- **Execution Time**: is the time interval between the time a service request arrives, and the time the corresponding response is generated. The execution time can be estimated by using existing performance estimation techniques, such as history data [29].

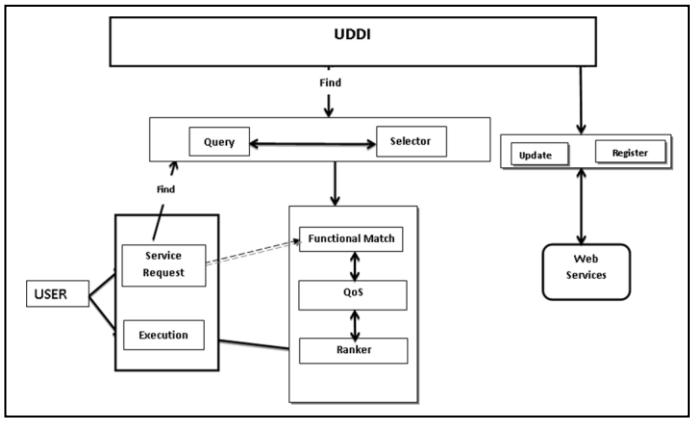

Figure 1. Web Service Selection Architecture

We assume that the response time can be predicted exactly, and simply make use of history data for a service provider´s computing power; in case of first time in using services we assume that processing speed standing for provider´s computing power.

- **Service Charge**: is the cost that the requester has to pay. The Web service cost can be estimated by operation or by volume of data. i.e. the monetary cost for request execution is commonly defined as:

$$C = Din\,.\,Pin + Texe\,.Pexe + Dout.\,Pout \quad (2)$$

Where, C indicates the total monetary cost, D_{in} represents the data volume transferred into the service provider, T_{exe} stands for the Request execution time, and D_{out} denotes the data volume transferred back to the user after request finish. P_{in}, P_{exe} and P_{out} indicate the prices for transferring in data, job execution and transferring out data, respectively.

- **Service Platform/API**: various platforms and APIs are provided for applications. The users may specify the platform or API requirements (e.g. the .Net applications, or Java/Spring based applications). If the application is originally developed on.Net platform, transplanting it to the Azure can offer tremendous savings in terms of time.

IV. System Architecture

The main focus of our approach is to design an intelligent system that has the potential of examining Web service's QoS properties in an open and transparent manner, and enabling clients to select the best available Web service by taking advantage of client QoS preferences, Web service capabilities, and service provider features. This is achieved through the following architecture of the proposed solution as shown on Fig. 1.

Web Services from different firms are stored on database using UDDI registry. The service selection algorithm is geographically separated and deployed on the Internet. It communicates with the database to find service providers.

A. Service request module:

This module uses a Web service ontology language (OWL-S) to communicate with the Query module to search for the Web service according to the functional service demand.

B. Services Response module:

This module presents the ranked set of services to the user also to collect users' appraisals towards the founded services. The collected information is offered for further action. by collecting the user's feedback and passing results to the QoS database for adjusting the service provider's appraisal dynamically according to the user's experience To guarantee the data quality, this module will change QoS attribute of user profile values dynamically after each user selection.

1) Query module:

This module communicates with the UDDI Registry to find all the service providers for user's request, and calculates some QoS values, e.g. response time. And store the result.

2) Selector module:

This module will return the service provider candidates; inside the selector we implement our provider selection algorithm.

3) Functional match module:

This module after receiving the services with QoS information , it filter the returned services by performing semantic match between the returned services and services request using OWL-MX matchmaker filters.

4) QoS module:

This module makes inquiries of QoS information regularly from a UDDI repository to check whether any Web service has added or withdrawn its QoS values. It changes information in the QoS database after the new Web service function has been classified that improves to a great extent the quality of the Web service discovery process. To continuously update services QoS values through UDDI, this module provides reliable service discovery results. In particular, it removes any outdated or broken links.

5) Ranker module:

This module uses the data collected from other modules to generate a ranked list of services. Inside the ranker we implement service filtering and ranking algorithm; using data warehousing techniques to provide decision about selection and ranking of required services by their QoS attributes as required by user. The detailed evaluation process will be discussed later.

6) Finally, Execution module:

This module is in charge of monitoring the execution state of the request. If the service provider is dead, it will be activated to find another service provider to execute the Request. After execution finished, it is also in charge of collecting the results.

V. Methodology of Web services selection

In this section we explore our proposed approach the proposed system proceeds as follow:

First, given a user request we search for candidate providers that can support this request and then we need to filter the resulted candidate services set generated from the SPSL (Service Provider Selection Algorithm) to

a) remove bad provider's, and

b) to decrease the search space; this is achieved by checking semantic matching between the candidates and service request.

Then, we perform functional matching (matching input/output parameters) using OWL-MX matchmaking algorithm [14] on the resulting services descriptions from running the SPSL algorithm.

The output of this step is thus a set of matched services with their QoS parameters. Next, we check user profile to get the weights of each QoS parameters and identify the expected user objective function towards the specified parameters. In case users' profile does not include those data we use his class assuming that each user class should contain an importance level towards QoS parameters. Then, we build a data cube whose dimensions are the QoS parameters of the functionally matched services with the aim of maximizing (or minimizing) their values according to the user's objective function. Building the data cube in our model acts as the ranking method for the services providers.

Consequently, the user is provided with a ranked candidate services list with an OLAP report about each service usage to enable him to make efficient decision in selecting the service that could provide the needed information.

Finally, we ask the user for feedback about the results to enhance future requests. In addition, the algorithm considers the case of 2 equivalent candidates; in this case we employ the user rating of those services to select one of them. In case no rating value is available, we provide the user with both services; and with the help of the resulting OLAP report he can decide which one fits his needs. In the following discussion we present the details of the algorithm.

A. Service Model

We assume in this work that services are of type request-response, i.e., they consist of one atomic activity (operation).

A service is represented as below:

Service tuple = <ID, I, O, provider_id, service_type, interface, processing_speed, price, trust_degree...&others>

where *ID* is the service identifier, *I* is the set of service inputs, *O* is the set of service outputs, *provider_id* uniquely identifies the provider; *Service type* indicates which kind of service this resource provides; we can access the service through *interface.*

Price indicates the monetary cost that users have to pay for resource utilization; *Trust_degree* represents the provider's reputation; We reserve the other criteria field to support more criteria.

B. Request Model

A service request is used by the framework to select a service provider for a single task from a set of services.

Request = < ReqID; UserID; Input_Data; Services type; Interface >

Where, *UserID* represents the requesters owner; *ReqID* uniquely identifies the request; *Input_Data* is the dataset that need to submit to the service provider; *service type* indicates which kind of service this request needs; and *interface* defines the platform and API that user prefers.

In our work the user request are divided in two parts the first part will be used to generate candidate providers that match specific services type then both the user input data and the discovered candidates are semantically annotated to perform semantic matching between them.

A service request is used by the framework to discover a set of services. Service request and existing services are both described based on common ontology mainly domain ontology and service ontology i.e. OWL-S [7], which allow service discovery and enable interoperability of the discovered services.

A service request consists of a set of semantic annotations (*ID, I, O*) that describe declaratively the desired service properties. *ID* is the request identifier, *I* is the service input, and *O* is the service output.

C. The Service Provider Selection Algorithm (SPSA):

After we select the set of providers that match the user request type as shown in Fig. 2, the number of candidate services is further reduced by checking semantic matching between the candidates and service request, and by adjusting user profiles as we will show in the next section.

D. Service Filtering and Ranking Algorithm

Given the results returned in Section V-C, we filter those candidate services by first performing semantic matchmaking between them and the requested services as follow:

Algorithm: Service Providers' Selection Algorithm
Input: user_id:requester, req_id:uniquely identifies the request ,
service_type:indicates which kind of service this request needs,
interface:defines the platform and API that the user prefers
Data: The service model of all services providers
Result: Service providers list (SPL)
if *The service provider is available & its service type and platform/API are the same as what's in the request required* **then**
  Search service providers from inside UDDI registry and return set of matched SPL list
end
forall the SP_i*(service provider)*$\in SPL$ **do**
  Calculate:{Price,Trust,Performance}
end
return *The SPL List* /* SPL contain the matched services with their provider QoS values */
if *No results are returned* **then**
  Let the user reconsider the values in his request and start again
end

Figure 2. The Service Provider Selection Algorithm

1) Functional matching.

Semantic Web based approaches have been applied to semantically annotate Web Services to allow automated discovery and ranking, followed by mediation and invocation. Rich semantic descriptions allow Service Providers to model their services in a more expressive way that makes it easier for the Service Consumers to search for the required service using semantic reasoning and querying approaches.

Use of semantic annotation is important for appropriate service discovery and to help user specify problem using free text which is translated to semantic description of the problem.

The users *Input_Data* are semantically annotated and also the candidate's services generated from SPSA. The functional part of a semantic Web service can be described by a quadruple SWS = (I , O, P, E), where I , O, P, E are sets of Inputs, Outputs, Preconditions, and Effects, with each parameter semantically annotated by means of an associated ontology "O". Matching a service request R with a service offer S is based on matching the individual parameters in the two descriptions. The service discovery process consists of checking all returned services from SPSA that semantically match the service request inputs, outputs (IO).

The proposed algorithm uses the OWLS-MX service matchmaker [30] to process service requests and advertisements described in OWL-S, and to compute the pair-wise similarities between parameters. In particular, we use this matchmaker because it provides five different matching filters. The first performs a purely logic-based match (M0), characterizing the result as exact, plug-in, subsumes, or subsumed-by. The remaining 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). 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 [30].

2) Non-functional and personalization.

After functional match is performed, we assume there are multiple sources of information for each service request; this implies that each request can be answered from multiple functionally similar Web services, so we need to decide which Web service provider is of higher quality. Hence, after services have been chosen based on functional parameters, non-functional matching is performed and user's profile has to perform another important task while using the service, it has to supply preferences of users towards the values of the parameters that are transferred to the service:

1. To save the time needed to identify the weight of different quality parameters each time user initializes a service request
2. To ensure providing the right service to the user as he expects.

We use user preference in our model assuming that each user profile contains three parts:

1) The first part contains the class of user (i.e. Business, economic, social … etc.).
2) The second part consists of the importance level of the preferred quality of service parameters and helps to choose between discovered services. We assume that these values range from 0 to 1.0. For example, a 0.99 value for price, and 0.44 for availability. Those values will be used as a user weight for QoS parameters. Also if the user considers all properties as important, then the weights are distributed equally. If user considers only certain attributes are important then the weights will be distributed equally between the other remaining attributes and also this part include the user preference for business properties of services like payment method.
3) The third part deals with the preferred characteristics of objects or information that a service claims to provide. This part helps to choose between different services and to discard services that can in principle handle the task, but do not provide any desirable objects. This is the user rating of service that can be 0 or 1.0.

In our study QoS constraint represents user's end-to-end QoS requirements. These can be expressed in terms of the user objective function towards the different QoS criteria as follows:

If the parameters values in a range less than 0.5, then we assume that the user wants to minimize the values, and if the values in a range greater than 0.5, then, we assume the user wants to maximize the objective function. Those preferences can be gathered from previous interactions in the form of a long-term profile or can be directly specified by the user in the form of soft constraints.

Definition 1: Given a candidate service set for a request denoted by *CSS*, and a vector *C* of QoS constraints on *CSS* given by*:* $C = c_1, c_2 \ldots c_m$. Let *S* be an instantiation of *CSS* in which a concrete Web service is selected. *S* is a feasible service selection iff *S* satisfies all QoS constraints in *C*.

In case of two identical services (i.e. two similar candidates) which candidate should be chosen is a crucial question. Deciding which one better presents the user's request when they are identical requires us to define an optimal service. Thus, it is important to address the problem of finding plans that consistently choose the highest quality available Web services.

Definition 2: An optimal service selection for a given Web service request *R*, and a given vector of QoS constraints *C* is a feasible selection with the maximum non-functional matching value. The non-functional matching value of each service is calculated by:

$$NF(S_i) = \sum_{i=1}^{d} W_i \times q_i(S) \qquad (3)$$

Where, *NF* (S_i) is the non-functional matching value, W_i is the weight for the QoS parameter identified by the user, and, q_i (*S*) is the value of the i^{th} QoS parameter in service *'S'*.

3) services ranking.

Next, we provide a ranking method to sort all matched candidate services based on user's preferences towards the criteria (time, cost, trust degree)

Rank=Functional Match value + non-functional match value. (4)

Non-functional match= $\sum_{i=1}^{d} W_i \times q_i(S)$

In case the users want to minimize the criteria value we multiple the value of non-functional match by (-1).

According to the research direction described in sections I, II we introduce a multi-dimensional user model in which the set of feasible services of user request are organized in an OLAP fashion, such that:

1. The cube dimensions represent the QoS parameters; We use the vector $Qs = q_1(s), \ldots, q_r(s)$ to represent the QoS attributes of service '*S*' which define our data cube dimensions, where the function $q_i(S)$ is the value of the i^{th} quality attribute of '*S*'.

In reality, companies managing the service searching engines can deploy special applications themselves to obtain their own experience on QoS of some specific Web services. Alternatively, they can also hire third party companies to do these QoS monitoring tasks for them [31].

2. The measure of each dimension is the value of Rank function.

Objective function towards those dimensions are calculated as we mentioned earlier depending on the weight for each value in the quality vector '*C*' the user identifies, to be used during the selection of services.

Note that, we map each item in the quality vector '*C*' into a single real value between 0 and 1, by comparing it with the minimum and maximum possible values of service candidates (for example, the maximum execution price of '*CSS*' can be normalized by selecting the execution price of the most expensive service in the candidate set) to allow a uniform measurement of the multi-dimensional service qualities independent of their units and ranges.

$$M(i) = \frac{q_i(S) - Q_{max}(i)}{Q_{max}(i) - Q_{min}(i)} \qquad (5)$$

Where, $M(i)$ is the normalization value for the i[th] dimension (for example "price"), $q_i(S)$ is the quality value of the i[th] dimension in Web service component '*S*' in the candidate set '*CSS*', $Q_{max}(i)$ is the maximum value of the i[th] dimension in the candidate set '*CSS*', and $Q_{min}(i)$ is the minimum value of the i[th] dimension in the candidate set '*CSS*'.

As shown in Fig. 3 each multi-dimensional entry on the model contains the objects needed to accomplish the non-functional selection and retrieving task of services; such objects are dynamically built at the beginning of the user sessions. In more detail, we exploit the OLAP logic model and use the well-known mechanism of aggregations on levels [32]. This allows us to represent and manage user variables with a very high level of granularity. The matched services are returned to user in an ordered list based on user objective function.

With the help of OLAP infrastructure we provide the user with the description and history of the services to decide which one to choose from. Service consumers might want to take a look at the history of these alternatives to make a better decision. If a service had good performance during the last year it might be more trust-worthy than services which have been recently published. This in turn is important to ensure result reliability that can help not only the user who wants to both save time and perform the right selection, but also large organizations that want to save money and time. Consequently, using data warehouse will benefit us in:

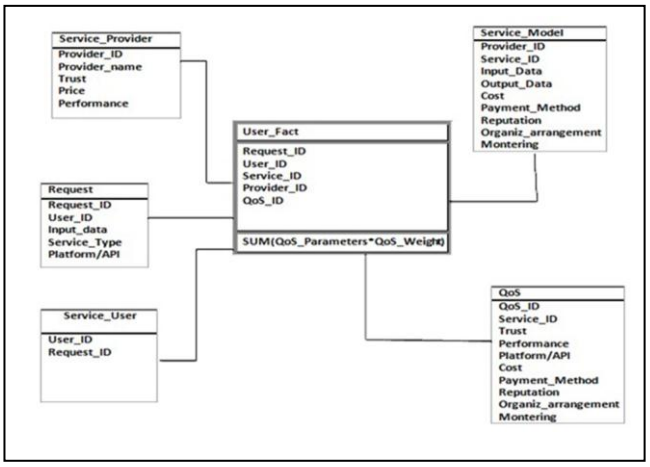

Figure 3. Star schema for non-functional and personalized selection of semantic Web service.

1) Past interactions are stored and can be retrieved to save time for search and selection of services.
2) Building a data cube with the selected QoS parameters (i.e. execution time, price, etc.), and objective function helps us to select and rank services that best match user request and preference in timely fashion.
3) If there are set of matched services that are functionally equivalent with different QoS, the model will find the best one of them based on user objective function and preference.
4) Each cube cell represents constraint over the selected service and so that the objective function to find best matched service can be performed. Also a ranked service list is provided.
5) The stored heuristic data about user interaction with different services are extremely useful for service providers as they can display reports about usage percentage of their services during specific periods of time.
6) Also, if the system is applied in a company the manager can identify that a specific service provider has poor rating and so does not accept his services in the future.

4) Update users profile.

After we found the service, we enable the user to provide rating on the provided services to show whether these services satisfy his needed information or not. The provided ratings are stored in the system for future use when the same or a similar request is issued.

5) Algorithm Overview.

In this section we explore some basic concepts and notations that we will use in our algorithm. Then, we present our proposed Service Filtering and Ranking algorithm (SFRA) shown on Fig. 4.

Definition 3. A Matched Solution List *MSoL* is a list of service providers that can serve the user request. It is similar, but not identical to the *SPL*:

a) MSoL is generated by filtering the SPL, by running OWL-MX.

b) Both the MSoL and SPL contain the service providers SP who will execute user request.

Definition 4. A Final Solution List *FSoL* is a list of service providers generated by filtering the *MSoL* such that: *FSoL* is generated by filtering *MSoL* based on the user profile values. The size of *FSoL* is smaller than the size of *MSoL*.

Definition 5. A Ranked Solution List *RSoL* is a list generated by sorting all the services in *FSoL* such that:

- *RSoL* contains exactly the same services in the *FSoL*.
- Service providers in *RSoL* are sorted using data cube.
- *RSoL* is returned to the end user.

VI. Experimental Results

In this section we discuss our experimental evaluation for our proposed algorithm. In our experiments we manually defined services. Services in the registry are all created and semantically annotated by humans (service developers). We assume in this scenario that all the services are meaningful. All services information's are stored in a data warehouse which is implemented by files updating instead of a real monitoring service, e.g. MDS service [33], also we used the open source for OWL-MX matchmaker [30] for functional matching of service as described in the functional matching Section.

Algorithm: Services Filtering and Ranking Algorithm(SFRA)
Input: R: User request, SoL Returned from Service Provider Selection Algorithm(SPSA)
Performs functional service matching using OWL-MX matchmaking filters on SoL to find matched candidate services($MSoL$) for the specified user request;
Groups all matched services with their QoS parameters and description;
forall the *QoS parameter* p **do**
  Retrieve the Max and Min value of p from the selected candidate ($MSoL$);
  /* The Max and Min values will be used in normalizing QoS values */
end
if *Existent* **then**
  Get the values of the user class and the importance levels (i.e. weight) of QoS parameters from user profile;
end
if *Request is found from the past interaction* **then**
  return services matched to this request
end
else
  Calculate the non-functional matching value of each service by:

$$F(ws_i) = \sum (W_j * Q(i,j))$$

  Add Functional and Non-Functional matching values to obtain a single cost function;
  Retrieve $FSoL$ list;
end
if *Equivalent services* **then**
  Add user rating value of service to $F(ws_i)$;
  Select Services with maximum $F(ws_i)$;
end
Build data cube for the $FSoL$ with the aim of maximizing or minimizing QoS values according to user objective function;
return $RSoL$ *the ranked results from the data warehouse with an* OLAP *report about each service usage;*
Let user evaluate the returned results to use this evaluation in the future;
if *No results are returned* **then**
  Let the user reconsider the values in his profile and start over again
end

Figure 4. Services Filtering and Ranking Algorithm

In the experiments we simulate different number of service providers ranging from 100 to 13000 providers, we use numeric code indicating the different types of service, uniformly ranging from 1 to 9. The price is evenly generated between 10 and 20 cent per service. And the trust degree is a decimal value uniformly generated from [0; 1].

The experiments were conducted on a DELL Inspiron 1525 machine with 2.80GHz processor Core Duo and 1 GB RAM. The machine is running under Microsoft windows 7 operating system.

In our experiments, we evaluate the performance of our proposed approach through two experiments; the scheduling latency under different number of services providers and different number of QoS criteria.

1. Scheduling latency under different number of services providers: Considering the QoS criteria: response time, monetary cost and trust degree, we evaluate the latency in responding to user request from different number of service providers. Fig. 5, shows that only a small number of service providers are available for the job under our configurations compared to SPSE [9].

Although, in Fig. 5, along with the number of service providers increasing from 500 to 3000, our algorithm has a much higher scheduling efficiency compared to the SPSE algorithm; our algorithm takes more time for semantic matching of services request as shown in Fig. 6.

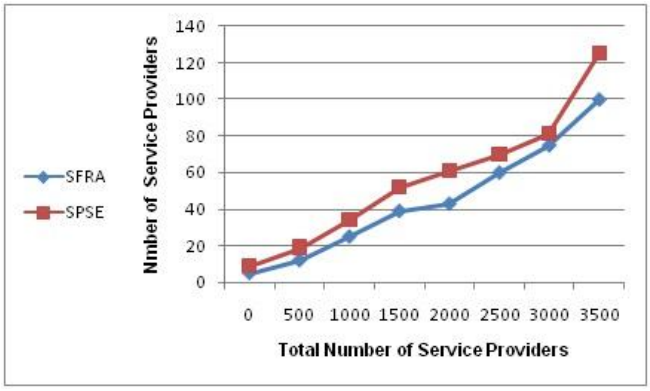

Figure 5. Available service candidates for the request.

For example when we have 2700 service provider our algorithm selects 75 that match the user request in 57:55 sec but SPSE algorithm selects 81 services in 38sec.

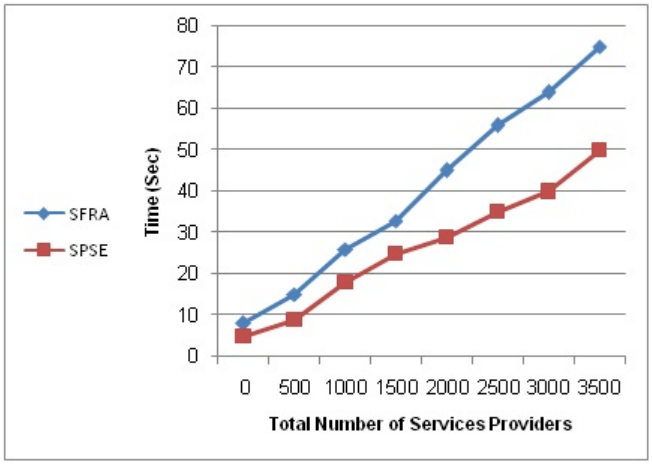

Figure 6. Latency for scheduling 150 request with different number of QoS criteria.

Therefore, our proposed algorithms SFRA is quite useful for user to accurately find the most appropriate service providers but with the cost of increasing the time.

2. The scheduling latency under different number of QoS criteria: Considering 1000 service providers and 150 service requests in the experiments, the SFRA and SPSE algorithms efficiency is also evaluated under different number of QoS criteria. Only the response time is considered in the first experiment, response time and monetary cost are integrated in to the second experiment, and response time, monetary cost and trust degree are implemented in the third experiment. As shown in Fig. 7, the time needed increases linearly with the number of criteria also SPSE algorithm take small time compared to our algorithm.

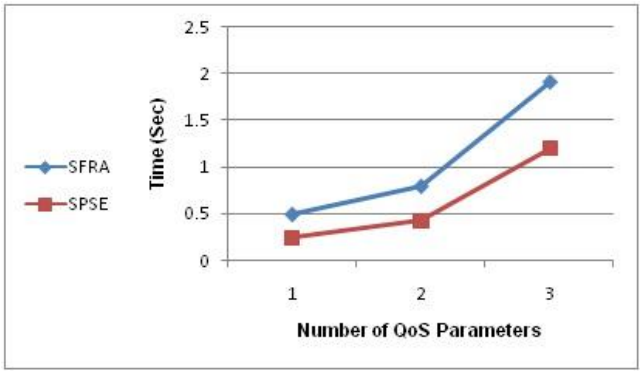

Figure 7. Comparison of SFRA and SPSE algorithms

We summarize the key findings of the comparison test between our approach and the SPSE engine as follows. The first observation is that an optimization of automated Web service discovery techniques appears to be necessary in order to assure the effectiveness of semantic Web services in larger search spaces of available Web services which can be expected in real-world scenarios. The second major outcome is that our approach can be considered as an optimization technique for automated Web Service selection because it can achieve significant improvements in computational performance. Our approach also assures scalability as it supports large number of service providers and QoS parameters, and it demonstrates a high accuracy among several invocations with marginal variations in the number of available services.

VII. Conclusions and Future Work

In this paper we propose an algorithm that is used as a tool in the selection of Web services based on the available providers and user requirements. Our algorithm basically selects the best set of services as required by the user. In addition, the proposed approach ranks those set of candidate services.

A key feature of our proposed approach is that, instead of asking the user for the non-functional properties, the algorithm uses the importance level for QoS parameters already stored in user profile, which makes the algorithm easy to use even for someone not very familiar with the different quality of services attributes.

Besides, the approach allows the user to rate any of the matched services, indicating how relevant or appropriate they are for his request. This rating is used to speed-up similar future searches.

For future work we aim to improve our Web service selection approach and allow it capturing maximum possible and relevant information from publicly available information in user's social network. Social aspects of the information from a network of service consumers and service providers can help a lot in ranking the best available Web services for the users. In addition, we want to consider the case where no matched services are found that match user QoS constraints. We want to explore the effect of relaxing those QoS constraints. Finally, we aim to investigate more on how we can take into account the users' opinion on the identified objective function to achieve better service matching.

References

[1] J. Rao and X. Su, "A survey of automated Web service composition methods," in Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, San Diego, California, USA, 2004, pp. 43–54.

[2] A. Averbakh, D. Krause, and D. Skoutas, "Exploiting user feedback to improve semantic Web service discovery," in ISWC 2009: Proceedings of the 8th International Semantic Web Conference. Springer-Verlag, 2009, pp. 33–48.

[3] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel, "Www or what is wrong with Web services discovery," in Proceedings of the Third European Conference on Web Services, ser. ECOWS '05. Washington, DC, USA: IEEE Computer Society, 2005.

[4] D. Martin, M. Burstein, G. Denker, J. Hobbs, L. Kagal, O. Lassila, D. McDermott, S. McIlraith, M. Paolucci, B. Parsia, T. Payne, M. Sabou, E. Sirin, M. Solanki, N. Srinivasan, and K. Sycara, "Bringing semantics to Web services: The OWL-S approach," in First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, CA, 2004, pp. 243–277.

[5] A. B.Bener, O. Volkan, and I. E. Savas, "Semantic matchmaker with precondition and effect matching using SWRL," Expert Syst. Appl., vol. 36, no. 5, pp. 9371–9377, 2009.

[6] L. Cabral, J. Domingue, E. Motta, T. Payne, and F. Hakimpour, "Approaches to semantic Web services: An overview and comparisons," The Semantic Web Research and Applications, vol. 3053, pp. 225–239, 2004.

[7] "Owl-s: Semantic markup for Web services," 2011.

[8] K. Pyar, "Decision support system for personnel information using data warehouse," in The 2nd Int. Conference on Computer and Automation Engineering (ICCAE). IEEE, 2010, pp. 668–672.

[9] L. Zhaoa, Y. Renc, M. Lib, and K. Sakuraia, "Flexible service selection with user-specific QoS support in service-oriented architecture," network and computer application, 2011.

[10] H. Q. Yu and S. Reiff-Marganiec, "Non-functional property based service selection: A survey and classification of approaches," NonFunctional Properties and Service Level Agreements in Service Oriented Computing Workshop colocated with The 6th IEEE European Conference on Web Services, vol. 411, pp. 13–25, 2008.

[11] D. A. Menasc, E. Casalicchio, and V. Dubey, "On optimal service selection in service oriented architectures," Performance Evaluation, vol. 67, no. 8, pp. 659–675, 2010.

[12] S. Agarwal and R. Studer, "Automatic matchmaking of Web services," in Proceedings of the IEEE International Conference on Web Services. IEEE Computer Society, 2006, pp. 45–54.

[13] H. Q. Yu and S. Reiff-Marganiec, "A backwards composition context based service selection approach for service composition," 2009 IEEE International Conference on Services Computing, pp. 419–426, 2009.

[14] C. Platzer and S. Dustdar, "A vector space search engine for Web services," in Proceedings of the Third European Conference on Web Services, ser. ECOWS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 62–71.

[15] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for Web services," in Proceedings of the Thirtieth international conference on Very large data bases, ser. VLDB '04. VLDB Endowment, 2004, pp. 372–383.

[16] S. Ran, "A model for Web services discovery with QoS," SIGecom Exch., vol. 4, pp. 1–10, March 2003.

[17] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng,"Quality driven Web services composition," in Proceedings of the 12th international conference on World Wide Web, ser. WWW '03. New York, NY, USA: ACM, 2003, pp. 411–421.

[18] D. A. Menasc´e, "QoS issues in Web services," IEEE Internet Computing, vol. 6, pp. 72–75, November 2002.

[19] D. Liu, Z. Shao, C. Yu, and G. Fan, "A heuristic qos-aware service selection approach to Web service composition," 2009 Eighth IEEEACIS International Conference on Computer and Information Science, pp.1184–1189, 2009.

[20] "UDDI version 3.0.2 specifications," http : ==uddi:xml:org=, 2011.

[21] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller, "A concept for QoS integration in Web services," in Proceedings of the Fourth international conference on Web information systems engineering workshops, ser. WISEW'03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 149–155.

[22] A. Sheth, J. Cardoso, J. Miller, and K. Kochut, "QoS for service-oriented middleware," in Proceedings of the Sixth World Multi-Conference on Systemics, Cybernetics and Informatics (SCI02), 2002, pp. 528–534.

[23] Y. Liu, A. H. Ngu, and L. Z. Zeng, "QoS computation and policing in dynamic Web service selection," in Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, ser. WWW Alt. '04. New York, NY, USA: ACM, 2004, pp. 66–73.

[24] N. Ibrahim, F. L. Mou¨el, and S. Fr´enot, "Mysim: a spontaneous service integration middleware for pervasive environments," in Proceedings of the 2009 international conference on Pervasive services, ser. ICPS '09. New York, NY, USA: ACM, 2009, pp. 1–10.

[25] E. Al-Masri and Q. H. Mahmoud, "Discovering the best Web service," in Proceedings of the 16th international conference on World Wide Web, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 1257–1258.

[26] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," ACM Trans. Web, vol. 1, May 2007.

[27] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for Web services composition," IEEE Trans. Softw. Eng., vol. 30, pp. 311–327, May 2004.

[28] D. A. Menasce, "Qos issues in Web services," IEEE Internet Computing, vol. 6, no. 6, pp. 72–75, 2002.

[29] S. hye Jang, V. Taylor, X. Wu, and M. Prajugo, "Performance predictionbased versus load-based site selection: quantifying the difference," in Proceedings of the 18th international conference on parallel and distributed computing systems, Las Vegas, Nevada, 2005.

[30] M. Klusch, B. Fries, and K. Sycara, "OWLS-MX: A hybrid semantic Web service matchmaker for OWL-S services," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 7, no. 2, pp. 121 –133, 2009.

[31] N. Ahmadi and W. Binder, "Flexible matching and ranking of Web service advertisements," in Proceedings of the 2nd workshop on Middleware for service oriented computing: held at the ACM/IFIP/USENIX International Middleware Conference, ser. MW4SOC '07. New York, NY, USA: ACM, 2007, pp. 30–35.

[32] W. H. Inmon, Building the Data Warehouse, 3rd Edition. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[33] "GT information services monitoring &discovery system(MDS)," http://www.globus.org/toolkit/mds/, 2011.