AN ARCHITECTURE FOR WEB SERVICE SIMILARITY EVALUATION BASED ON THEIR FUNCTIONAL AND QOS ASPECTS

Mahsa Jamal Vishkaei[1], Ahmad Baraani-Dastjerdi and Kamal Jamshidi[2]

[1]Department of Computer Engineering, University of Sheikhbahaee, Isfahan, Iran
vishkaei@shbu.ac.ir
[2] Department of Computer Engineering, University of Isfahan, Isfahan, Iran
{ahmadb, jamshidi}@eng.ui.ac.ir

ABSTRACT

By increasing popularity of SOC, using Web services in applications has increased too. SOC creates a loosely coupled environment in which the actual execution environment might differ significantly from the one with the presupposed conditions during application design. Therefore, although an appropriate Web service might have been selected, by passing time, the Web service may not be efficient enough or may not be applicable under specific conditions.

For service-oriented systems to be flexible and self-adaptive, it is necessary to automatically select and use a similar service instead of the one which causes the above mentioned problems. Finding a similar service means specifying the proper services which fulfill the same requirements as those fulfilled by the problematic service.

In most of the previous works, a number of the best services (k) are selected and ordered based on functional similarity. The user must select one of these services based on his/her preferences. One important metric in selecting a similar service is considering QoS properties and user preferences about QoS. Because of the importance of this issue, in the present paper, an architecture is proposed in which, in addition to functional similarity, QoS properties and user preferences are also considered in selecting a similar service.

KEYWORDS

Web service, Self-adaptive, Functional Similarity, QoS Similarity &User Preferences

1. INTRODUCTION

"SOC promotes the idea of assembling application components into a network of services that can be loosely coupled [1] and Web services are currently the most promising SOC based technology [2]. Web services act dynamically in such an environment and therefore, there could be real-time changes in service status such as service unavailability and service quality decline. Such problems may reduce quality or cause failure in processes and applications which use such services. This makes the service consumer to go through the process of rediscovering a service similar to the initial one which could also fulfill the previous requirements. Such a process is much time-consuming. A flexible and self-adaptive Service-oriented system must be able to automatically select the similar services and introduce them to the user so that the user does not have to go through the difficulties of discovering similar services. The present work offers a solution in providing similar services automatically whenever there is a problem in initial service availability. Similar services are considered those which have a close functionality and QoS to the initial one. In the process of finding similar services, after finding some services which have the most functional similarity, the important metric for the user is to select the service which has a satisfactory level of QoS. For Web services users, considering QoS issues is

DOI : 10.5121/ijwsc.2011.2201

critical since there is a direct relationship between the quality of an application consisted of Web services and the quality of each consisting service. Thus, finding a similar service does not only encompass considering functional features, but also QoS related properties. For this purpose, there is a need to seek a way to know the user's preferences about QoS. In most studies such as [3,4,5,6], finding similar services is based on functional similarity in which a number of the best services (k) are selected and introduced to the user. The user then has to select one of them based on his/her preferences about QoS.

The represented method in this paper, considers QoS properties and user preferences about these properties in addition to functional similarity. Considering QoS properties results in a different rating of functionally similar services and, as a result, the best possible selection is done based on functionality and quality.

Using QoS properties, results in a selection based on another important aspect of services which optimizes service selection. In case of any changes in QoS properties of services, the system adapts itself to environmental conditions and automatically selects the best similar service. To gain service quality information, a four layered architecture is introduced in this article which monitors services and stores this information for future use. When there is a request to find a similar service, the first step is to examine services based on functional similarity. The functionally similar services are then examined based on quality and user preferences. At last, services are rated based on all the above similarity metrics. Accuracy is increased by using statistical methods. In addition, each functional and QoS similarity has a weight which could be changed based on user's opinion and environmental conditions which makes the final decision flexible.

User QoS preferences are derived using SLA (Service Level Agreements). SLA is a commonly used mechanism to express Quality features [7]. In the present work, the attempt is to introduce a new method in which: first, using SLA, user-defined parameters and their values are derived and used automatically after discovering functionally similar services; second, the final decision is flexible based on functionality and quality metrics. Thus, the present study attempts to find a similar service based on two aspects.

The paper is organized as follows: Section 2 introduces related works. Section 3 explains the QoS model that refers to QoS properties which used for quality evaluation of service. Next, Section 4, presents our Architecture for similarity evaluation in detail. Finally we get conclusion in section 5.

2. RELATED WORKS

Similarity search for Web services, also called Web service retrieval, occupies an important place in SOC and several related works could be found regarding the issue. Generally, there are three major groups of methods for finding similar services. In the first group, there exists a group of previously chosen similar services; when a service fails to work at runtime, it is replaced by another based on user context or QoS [8,9]. In the second group, similar services are selected dynamically [3,4,5,6]. In the third group, the external behavior of a Web service like execution paths or its conversations with other services is considered. In this group, because of lack of information about external behavior of services in their description, service check is done in composition process [10,11,12,13,14].

The second group is considered basic for this article. The reason is, the methods in this group base their work on information existent in service description (WSDL) rather than concerning external behavior or defining a new model for service representation or even choosing similar services in advance. In works [3,4,5,6], calculating similarity is based on functional aspects only

and therefore, the user needs to do further refinement pertaining to important QoS features. In [3], both syntactic and semantic aspects of a Web service that could be derived from WSDL are considered. Semantic aspects are related to the purpose of a Web service which is in turn related to the names assigned to the entire service and syntactic aspects are based on input/output structures and data type adaptations. In [4], a search engine named Woogle is established for Web services which uses textual similarity of methods and its parameters in order to examine service similarity. The key element of Woogle is clustering algorithm for identifying the relationships among the terms adopted in the all published Web services. It then compares the concepts encompassing input/output parameters as a measure of similarity. In [5], finding similar services is based on domain-independent and domain-specific ontology. In order to specify domain-independent relations, after a series of pre-processes, WordNet thesaurus is used. Deeper relations based on industry and application-specific terms are found using domain-specific ontology and after that, related terms are found based on rule based inference. Matches due to the two methods are combined to determine an overall similarity score. In [6], because of inefficiency of catalogue style service discovery methods, a new method has been developed in which similarity is sought via comparing the two WDSLs. In this article, in order to find the similarity between two WSDL descriptions, a series of complementary methods are introduced. These methods examine, on the one hand, data type structures, messages and operations and, on the other hand, the meaning of identifiers and natural language descriptions. These methods combine classical information retrieval and WordNet-based technique to increasing the precision of the retrieval mechanism.

From the first group, work [15] could be mentioned in which, the assumption is that there is a series of functionally similar services from which, one service is selected based on QoS. It uses preferences networks to represent user preferences and to decide upon QoS using such preferences. The work does not mention how to obtain user preferences but indicates that these preferences can be defined at three levels of low, medium and high. Such a definition cannot be accurate enough since different people may have different conceptions of these three levels.

3. THE QOS MODEL

The term “QoS” was used for the first time in the networking community by Crawley [16]. In SOC, QoS encompasses a number of qualities or service properties like availability, security, response time and throughput [17]. Generally speaking, QoS attributes are divided into two groups: deterministic and non-deterministic [18]. Deterministic attributes are those that their value is known before a service is invoked, like price or supported security protocols. Non-deterministic attributes are those which their accurate value is unknown until the service is invoked, like response time.

In this section, some QoS attributes are introduced which are used to evaluate the extent of similarity among Web services from quality point of view. These features are defined under specific conditions, for example they must be measurable, being measurable means that they could be measured through monitoring mechanisms, to name the most important. Stated simply, the purpose here is to use non-deterministic features. This results in a real evaluation of services in operational environment and thus has an important role in finding similar services. To create a general open model for evaluating QoS, there is also a need to consider features with a high percentage of generality among QoS features of Web services which their desired value is mentioned in SLA so that user preferences are discovered automatically. In this article, those features used to measure QoS similarity are called “metrics”. These metrics include Availability (A) and Response time (R). It is also possible to add other features later.

4. THE ARCHITECTURE

The architecture proposed in this section finds similar services to the initial service (Sq) based on functional and QoS similarity. This architecture is composed of four layers (Figure 1).

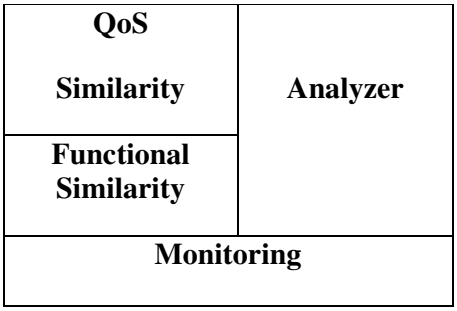

Figure 1: Architecture

The monitoring layer monitors Web services in the service repository ($\sum = \{S_p\}$) and stores obtained data in a Database. In functional similarity and QoS similarity layers, functional and QoS similarity of the Web services in the repository are evaluated compared to Sq. The analyzer layer coordinates all the layers and makes the final decision. This layer communicates with the external user and receives requests to find similar services and sends the final answer to the user.

In this architecture, functional similarity is examined through WSDL. Services are examined for QoS through monitoring QoS metrics of all services in repository and storing obtained data. This is followed by evaluating QoS similarity of monitored services with user specified QoS metrics related to S_q through the specification of user preferences about QoS metrics. An examination of the stored information is done through monitoring operation and the degree of similarity between QoS metrics of services with user preferences is identified. Not all services need to be checked at this stage. Only those services whose functional similarity is greater than a defined threshold are examined. Finally, services are rated based on the degree of similarity obtained from two different aspects. Furthermore, this rating is done in a flexible manner and thus, the best possible similar services are found and offered to the user. The component diagram of the architecture is presented in Figure 2.

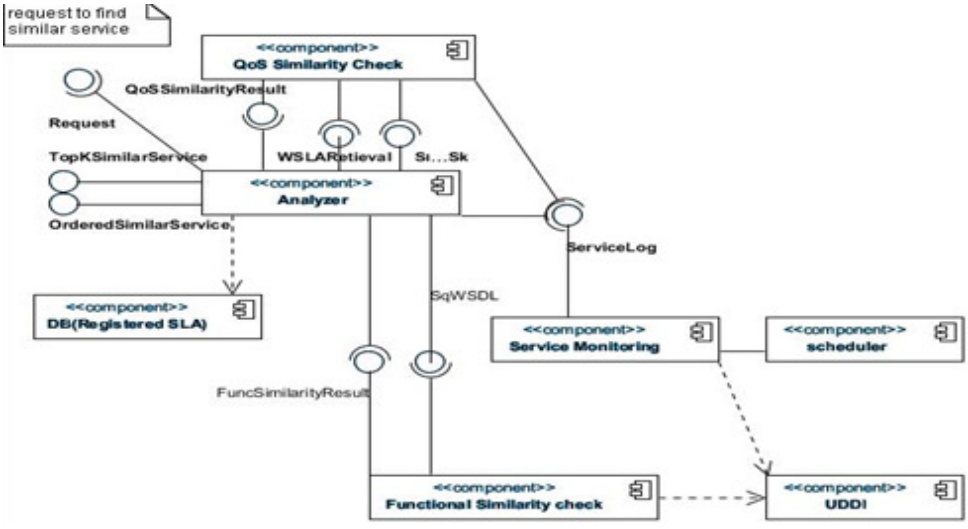

Figure 2: Component Diagram

Details about each layer are presented in the following sections.

4.1. The monitoring layer

The monitoring layer identifies and stores the QoS information of Web services. One of the problems of the current Web services is their QoS information not being mentioned in their description [17,19]. As a result, there is a need to find a way to monitor services and get such information dynamically so that it could be used in future.

To get the required QoS information, the method for monitoring services in the repository $\sum = \{S_p\}$, must:

1. Have the ability to get the required information using Web service description (WSDL), since, the code and implementation of the service is generally invisible to users;
2. Not need to do any change to the Web service;
3. Be independent form Web service provider and be applicable to all Web services.

In most works about QoS in Web services, the way to get and evaluate these features is not mentioned; for example, in [20], the UDDI repository for associate QoS to specific Web service is extend without any mentioning of how such values were obtained. In [21], analyzing and estimating the performance of Web services is based on simulation i,e, invoking a Web service under low load conditions and transforms these testing results into simulation model and uses the model to estimate service excepted performance in heavy load. Because Web services act dynamically, it does not seem that methods based on estimation be much accurate. [22] also proposes a framework for QoS monitoring and analysis. This work considers communication level monitoring via SOAP messages interception but has not detail about it and is mostly concerned with analyzing the information. In [23] selecting services is based on QoS and it tries to integrate QoS into Web service technology. But again in this work nothing is mentioned about the way to get and evaluate QoS attributes.

After studding the existing methods and the above mentioned requirements, the method in [24] was found suitable. This method is Non-intrusive, it measures QoS properties dynamically and in a bootstrapping way and, in addition, completely service independent and does not have access to Web service implementation. The measurement technique in this method is client-side which is independent from the service itself and the service provider. In client-side technique it is enough to have access to Web service description to get the QoS features of the service while server-side technique need to access the Web service' source code. Based on what mentioned before, the latter is not a suitable technique here. [24] uses aspect-oriented programming (AOP) which allows weaving performance measurement aspects. Thus, this approach could be used as an independent package for monitoring services and recording the required information. Availability (A) and response time (R) metrics of services could also be measured using this method. It is noticeable that using this method should be so that extra loads are not imposed on services. If all services are monitored all the time, a huge amount of information must be stored; in addition, extra loads may be forced on services. To prevent this, it is necessary to reduce the amount of data without distorting its integrity. This is achieved by sampling in monitoring and data storage. Services are monitored randomly or in static time intervals. A scheduler component, in which scheduling policies are defined, is in charge of sampling. This process is continued by collecting the measured features for each service and storing the data in a Data base and using this data when necessary. When QoS data are collected, it needs to be processed to fulfill its particular purpose. The processing of data could be online or offline or a combination of both [25]. In online processing, the data are processed immediately and in offline processing the data are processed after being stored. Offline processing has the advantage that the data could be studied from various viewpoints. In the present work, based on the objectives of the study and the defined usage for the data, offline processing was preferred.

4.2. The functional similarity layer

The functional similarity layer checks the degree of functional similarity between services in repository ($\sum = \{S_p\}$) and Sq. Checking functional similarity means finding those services that do the similar task to service Sq. The main source to be used here is WSDL description. The required information can be obtained from main parts of the WSDL, i.e portType, operation and message. After receiving the WSDL of Sq, its similarity to the services in the repository ($\sum = \{S_p\}$) is measured and each service is rated based on its functional similarity. Those services that their degree of similarity is higher than the threshold are chosen and named as services $S_1..S_k$. In the next step, the vector of $F = (fs_1, fs_2, \ldots, fs_k)$ is created for services $S_1.. S_k$ from their functional similarity. Services $S_1..S_k$ and vector F are then sent to the analyzer layer.

As mentioned in section 2, in works [3,4,5,6] the similarity between two Web services is measured from functional point of view. In this section, one of these methods is selected for evaluating functional similarity between Web services as follows.

In [4], terms are considered as a package of words and similarity is measured based on TF/IDF measure, the concepts are inferred from terms and the similarity among these concepts is noticed. The weakness of this work is that it is possible to send only one method to the Web service. In [5] the focus is only on words and the structure of the WSDL is not considered which is the weakness of this method. [6], like [3], uses a recursive method in measuring similarity between service description elements but its weakness is not considering the number of operations and parameters of Web services. Work [3] does not have the above mentioned problems and is accurate enough; therefore it is used in the present work to measure functional similarity. The latter method considers both syntactic and semantic aspects of Web services that could be derived from WSDL. Semantic aspects are related to the purpose of the Web service which is itself related to the names assigned to the entire service like the names of operations, parameters, port types, parts and inputs and outputs of its methods. Syntactic aspects are related to the conformance between input and output structures and the consistency among data types.

4.3. The QoS similarity layer

The QoS similarity layer measures the degree of QoS similarity of services in repository $\sum = \{S_p\}$ to S_q. Achieving this goal requires calculating the vector of user preferences ($P_{uq} = (a_{uq},r_{uq})$) about service QoS features for S_q in which a_{uq} indicates availability and r_{uq} indicates response time. The next step is to evaluate the quality status of services using the information calculated and stored by the monitoring layer. It is noticeable that only those services which are functionally similar to S_q are examined here. In section 4.3.1. how to calculate P_{uq} and in section 4.3.2. how to measure QoS similarity are discussed.

4.3.1. User preferences about QoS

For Web service users, considering quality issues are very important because the quality of applications consisting of Web services has a direct relationship with the quality of each service. Therefore, there is a need to calculate P_{uq}. One method is using SLA. By using SLA, one can automatically become aware of user preferences when choosing S_q and use them in finding similar services. SLA is actually a kind of contract in which different metrics for quality is defined [17]; for example, the average response time should be less than 0.5 second or the availability of a service must be more than 99.0 %.

In order to use SLA, it is necessary to use one of its defined standards. One standard is WSLA (Web Service Level Agreement) [7] that is a formal language for expressing SLA in which the agreement is made at service level.

The basic parts of a WSLA are as follows [7]:

1. Parties and their roles: provider, consumer and third parties;
2. SLA parameters: service object specifications like response time, throughput, etc. ;
3. Service Level Objectives (SLO): promises made about SLA parameters, obligations of each party and actions taken if these promises and obligations are not observed.

It is obvious that in order to realize user preferences and to make the P_{uq} vector about service specifications, one must use the third part of WSLA i.e. SLO. In WSLA, it is possible to define arbitrary parameters. It is also possible to have different definitions for the same parameter like availability. In order for this article to be comprehensive, for each parameter, only one definition is used and in all WSLAs for different services it is interpreted the same. In order to understand better, notice a sample SLO in Figure 3.

```
<ServiceLevelObjective name="Conditional SLO For AvgThroughput">
<Obliged>ACMEProvider</Obliged>
</Validity>
        <Expression>
        <Implies>
                <Expression>
                <Predicate xsi:type="Less">
                <SLAParameter> Response Time</SLAParameter>
                <Value>10</Value>
                </Predicate>
                </Expression>                                   } part 1
                <Expression>
                <Predicate xsi:type="Greater">
                <SLAParameter>AvgThroughput</SLAParameter>
                <Value>1000</Value>
                </Predicate>
                </Expression>
        </Implies>
        </Expression>
<EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Figure 3. A sample SLO

As is seen in Figure 3, part 1 shows the extent considered for QoS parameters that could be used to find the most similar service in QoS to the initial one. For each attribute, it is specified that the desired value must be greater or lower than the mentioned number. For example, for the average throughput, a number greater than 1000 and for response time, a number less than 10 is specified. This is how the vector of P_{uq} for WSLA concerning S_q is created.

In WSLA, it is possible to define parameters at both method-level and service-level. In this work, the assumption is that parameters are defined at service-level and in addition the WSLA between service provider and service consumer for each Web service is stored in Database.

4.3.2. Evaluating QoS similarity

In this section, examining services from QoS point of view is discussed. In order to evaluate QoS similarity of services with user preferences about Sq, it is necessary to communicate with the analyzer layer. Through this communication, services $S_1..S_k$ and the WSLA of S_q ($WSLA_{sq}$) are received and the QoS similarity of services that are functionally similar to S_q are evaluated. The vector of P_{uq} is filled with the average availability and the average response time values from $WSLA_{sq}$. To evaluate the degree of QoS similarity, it is also necessary to use the data stored for services $S_1..S_k$ by the monitoring layer. The average availability (a_{sj}) and the average response time (r_{sj}) for services $S_1..S_k$ are calculated using the stored data and put into matrix M (Figure 4). In recovering the monitored data and calculating the average availability and the average response time a number of recently stored data (w) are used. The purpose is considering the most recent service behavior so that if the service has been acting well previously but not recently, such a fact makes a difference in decision making and at the end the best possible selection is done.

$$M = \begin{matrix} A & R \\ \end{matrix} \begin{bmatrix} a_{s1} & r_{s1} \\ \vdots & \vdots \\ a_{sk} & r_{sk} \end{bmatrix} \qquad (1)$$

$$where \quad a_{sj} = \frac{1}{w}\sum_{x=n-w}^{n} a_{sjx} \quad , \quad r_{sj} = \frac{1}{w}\sum_{x=n-w}^{n} r_{sjx} \quad , \quad \forall j = 1...k$$

a_{sjx} : X[th] stored data for service[j] , n : total stored data , w : number of recently stored data

a_{sj} : average availability for service[j] , r_{sj} : average response time for service[j]

Figure 4

Calculating the similarity of vector P_{uq} and matrix M is actually a calculation in Euclidean space in which P_{uq} and each element of M are like points in space with two dimensions of A and R. It is noticeable that data in P_{uq} specifies the two desirable thresholds for availability and response time from user's point of view; this means that the user prefers service availability be greater than a_{uq} and service response time be less than r_{uq}; the more difference between these two, the more satisfied the user. Therefore, Euclidean distance could be used to calculate similarity between P_{uq} and M.

In using Euclidean space, if there is great difference among data values or there is a difference in measurement units of specifications, it is necessary to normalize the data; this assures assigning the same weight to all specifications [26]. Here, because of the difference between the measurement scales of availability and response time, P_{uq} and M data must be normalized. The normalization is done using the min-max relation [26], formula 2. For example, if the minimum and the maximum values for A are min_A and max_A respectively, and (a) is the old value of A, based on formula 2, the new value of A, in the new range, (new_min_A, new_max_A), is a'.

$$a` = \frac{a - \min_A}{\max_A - \min_A}(new\_\max_A - new\_\min_A) + new\_\min_A \qquad (2)$$

For each element of P_{uq} and M, formula 2 is used to create P'_{uq} and M' (Figure 5). Here, the new range is [0, 1].

$$P`_{uq} = (a`_{uq}, r`_{uq}) \quad , \quad M` = \begin{bmatrix} a`_{s1} & r`_{s1} \\ \vdots & \vdots \\ a`_{sk} & r`_{sk} \end{bmatrix}$$

$$\text{where } a`_{sj} = \frac{a_{sj} - a_1}{a_2 - a_1} \quad , \quad r`_{sj} = \frac{r_{sj} - r_1}{r_2 - r_1} \quad , \quad \forall j = 1..k \wedge uq$$

$$a_1 = \min\{a_{uq}, a_{s1}, a_2, \cdots, a_{sk}\}, \quad r_1 = \min\{r_{uq}, r_{s1}, r_2, \cdots, r_{sk}\}$$

$$a_2 = \max\{a_{uq}, a_{s1}, a_2, \cdots, a_{sk}\}, \quad r_2 = \max\{r_{uq}, r_{s1}, r_2, \cdots, r_{sk}\}$$

Figure 5

The degree of similarity can now be calculated using Euclidean distance. Calculating similarity is done using QSim in Formula 3 and the answer is stored in Q vector (Figure 6).

$$\begin{cases} Q[j] = QSim\left(P`_{uq}, M`[j]\right) = \sqrt{a^2 + r^2} \quad , & \forall j = 1..k \\ where \begin{cases} a = (a`_{uq} - a`_{sj}) \quad if\,(a`_{uq} \le a`_{sj})\ else \quad a = 0 \\ r = (r`_{uq} - r`_{sj}) \quad if\,(r`_{uq} \ge r`_{sj})\ else \quad r = 0 \end{cases} \end{cases} \tag{3}$$

Figure 6

4.3.3. Optimization

Calculating the similarity between P'_{uq} and M' cannot be only based on average availability and average response time since high data variation from these two may affect accuracy. Therefore, in order to increase accuracy, it is necessary to consider the degree of variation from average as well. Thus, in calculating similarity between P'_{uq} and M', coefficient of variation (CV) is used. Low CV shows consistency among data and high CV shows inconsistency among them [27]. Using data in (or By having a set of data objects) $X = \{x_1, x_2, \ldots, x_n\}$, CV is calculated through formula 4 [27] :

$$CV = \frac{s}{\bar{x}} \quad where \quad \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \quad , \quad s = \sqrt{\frac{\sum_{i=1}^{n} \left(x_i - \bar{x} \right)^2}{n-1}} \tag{4}$$

Therefore, in order to increase accuracy in calculating QoS similarity, the CV value for each of the availability metric (cv_a) and response time metric (cv_r) is calculated for services S_1.. S_k from data stored in monitoring through the time span previously mentioned. These numbers are put into QSim (P'_{uq},M'[j]) in (3) and (5) is created:

$$Q[j] = QSim(P`_{uq}, M`[j]) = \sqrt{\frac{1}{cv_a} a^2 + \frac{1}{cv_r} r^2} \quad ; \forall j = 1..k \tag{5}$$

The Q vector shows the degree of similarity of each service to P'_{uq}. To increase accuracy, the assumption is that cv_a and cv_r are less than one. It is possible that any element of Q be out of [0,1] range, therefore it is necessary to put them back in the boundary using (2). The new vector is named Q' and its elements are named as qs_j; thus Q' is represented as $Q'(qs_1, qs_2, \ldots, qs_k)$. Now the final decision is made using Q' and the results of functional similarity evaluation.

4.4. The analyzer layer

The analyzer layer is responsible for coordinating all the layers and producing the final result. This layer communicates with external user and receives requests for finding similar services and sends the final answer to the user. When a request to find similar services to S_q is received, the analyzer sends the WSDL of S_q to the functional similarity layer, which checks for functional similarity. The result is a list of services $S_1..S_k$ together with their degree of similarity to S_q which is sent back to the analyzer. Notice that this result has the form of $F = (fs_1, fs_2, \ldots, fs_k)$. The analyzer then sends the list of services ($S_1..S_k$) to the QoS similarity layer, which has to check for QoS similarity. It also sends the specific WSLA based on the requesting party, the provider and S_q. the QoS similarity layer produces the result in the form of Q' in which the extent of QoS similarity of services $S_1 \ldots S_k$ to P_{uq} is presented. In the analyzer layer

the overall similarity of services $S_1 \ldots S_k$ and S_q is calculated by creating a matrix of S_{kx2}whose columns are filled with functional and QoS similarity values previously calculated (Figure 7).

$$S = \begin{bmatrix} fs_1 & qs_1 \\ fs_2 & qs_2 \\ \vdots & \vdots \\ fs_k & qs_k \end{bmatrix}$$

Figure 7

Finally, the overall similarity of services $S_1 \ldots S_k$ to S_q is calculated in the analyzer layer as follows and services are ranked and ordered. The overall similarity means both functional and QoS similarity at the same time.

4.4.1. Calculating the overall similarity

In order to calculate the overall similarity and ranking services, it is necessary to consider functional and QoS similarities and their degree of importance. Therefore, in analyzer, a weight is assigned to functional and QoS similarities. This weight is applied through $W = [w_1, w_2]$ where w_1 stands for functional similarity and w_2 stands for QoS similarity and $w_1+w_2=1$. w_1 is always greater than w_2 because the purpose is to find a service which does the same work with good quality. Of course these weights could be changed based on the type of work and user's opinion. Total similarity ranking of services $S_1 \ldots S_k$ to S_q is done using (6).

$$A^{score} = \begin{bmatrix} fs_1 & qs_1 \\ fs_2 & qs_2 \\ \vdots & \vdots \\ fs_k & qs_k \end{bmatrix} \times [w_1, w_2] \qquad (6)$$

Each element of A^{score} is calculated based on (7).

$$A_j^{score} = w_1 \times fs_j + w_2 \times qs_j \quad , \forall j = i..k \qquad (7)$$

Based on total similarity rank, A_j^{score} , services are ranked and ordered. The service with the highest value of A_j^{score} is the most similar and its rank is 'first'; similarly, a list of ranked services based on A_j^{score} is created and sent to the user.

5. CONCLUSION

There could be real-time changes in service status such as service unavailability and service quality decline in SOC environment. For service-oriented systems to be flexible and self-adaptive, it is necessary to automatically select and use a similar service instead of the one which causes problems and introduce them to the user so that the user does not have to go through the difficulties of discovering similar services. The present work offers a solution in providing similar services automatically whenever there is a problem in initial service availability.

One important metric in selecting a similar service is considering QoS properties and user preferences about QoS. Because of the importance of this issue, in this work, an architecture is proposed in which, additional to functional similarity, QoS properties and user preferences are also considered in selecting a similar service.

In our architecture to check functional similarity, WSDL of services is used. Checking functional similarity means finding those services that do the similar task. To check QoS similarity, all services in the repository are monitored and the results are stored in a Database. After automatically obtaining user preferences about QoS, QoS similarity of services to user preferences is checked. In order to increase accuracy in QoS similarity check, statistical methods are used. Total similarity is calculated based on functional and QoS similarity in a flexible way. Considering QoS properties results in a different rating of functionally similar services and, as a result, the best possible selection is done based on functionality and quality.

In future works, the objective is to extend QoS model with deterministic parameters like cost, security, etc.

REFERENCES

[1] Michael P. Papazoglou,Paolo Traverso,Schahram Dustdar,Frank Leymann, (2007) "Service-Oriented Computing: State of the Art and Research Challenges ," *IEEE* , vol. 40, no. 11, pp. 38-45.

[2] S. Weerawarana, Ed., (2005) *Web Services Platform Architecture:SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable*. Prentice Hall.

[3] P.Plebani,B.Pernici, (2009) "URBE: Web Service Retrieval Based on Similarity Evaluation," IEEE Transactions on Knowledge and data engineering, vol. 21, no. 11, pp .1629-1642.

[4] X.Dong,A.Y. Halevy,J.Madhavan,E.Nemes,J.Zhang, (2004) "Simlarity Search for Web Services," in Thirtieth international conference on Very large data bases, vol. 30, pp. 372-383.

[5] T.S.Mahmood,G.Shah,R.Akkiraju,A.A.Ivan,R.Goodwin, (2005) "Searching Service Repositories by Combining Semantic and Ontological Matching," in IEEE International Conference on Web Services (ICWS '05), pp. 13-20.

[6] E.Stroulia,Y.Wang, (2005)"Structural and Semantic Matching for Assessing Web-Service Similarity," International Journal of Cooperative Information Systems, vol. 14, no. 4, pp. 407-438.

[7] "Web Service Level Agreement (WSLA) Language Specification," IBM, 2003.

[8] Y.Taher, D.Benslimane, M.C. Fauvet, Z. Maamar, (2006) "Towards an Approach for Web services Substitution," in in 10th International Database Engineering and Applications Symposium IEEE, pp. 166-173.

[9] Y.Yamato, H.Sunaga, (2007) "Context-Aware Service Composition and Component Change-over using Semantic Web Techniques," in IEEE International Conference on Web Services, pp. 687-694.

[10] M. Mecella,B. Pernici,P. Craca, (2001) "Compatibility of E-Services in a Cooperative Multi-Platform Environment," in Int'l Workshop Technologies for E-Services (TES '01), pp. 44-57.

[11] g.spanoudakis,A. Zisman, A. Kozlenkov, (2005) "A Service Discovery Framework for Service Centric Systems," in IEEE Int'l Conf. Services Computing (SCC '05), pp. 251-259.

[12] L.Kuang, (2008) "A Formal Analysis of Behavioral Equivalence for Web Services," in IEEE Congress on Services, pp. 265-268.

[13] P.Cyrille H´eam, O. Kouchnarenko , J.o.Voinot, (2007)"How to Handle QoS Aspects in Web Services Substitutivity Verification," in 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp. 333-338.

[14] A. Martens, (2005) "Process Oriented Discovery of Business Partners," in Enterprise Information

Systems (ICEIS '05), pp. 57-64.

[15] G.Ram, Santhanam,S.Basu,V.Honavar, (2009) "Web Service Substitution Based on Preferences Over Non-functional Attributes," in International Conference on Services Computing (SCC 2009), Bangalore, pp. 21-25.

[16] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A Framework for QoS based Routing in the Internet".

[17] A.Menasce, Danial, (2002) "QoS issues in Web services," in IEEE Internet computing, pp. 72-75.

[18] Y. Liu, A. H. Ngu, and L. Zeng, (2004) "QoS Computation and Policing in DynamicWeb Service," in Proceedings of the 13th International Conference onWorldWideWeb (WWW'04)ACM Press, New York, NY, USA, pp. 66-73.

[19] A.Mani, A.Nagarajan, (2002) "Understanding quality of service for Web services," IBM http://www-128.ibm.com/developerworks/library/ws-quality.html.

[20] S. RAN, (2003) "A Model for Web Services Discovery With QoS," ACM SIGecom Exchanges, vol. 4, no. 1, pp. 1-10, Nov.

[21] H.G. Song, K.Lee, (2005)"sPAC (Web Services Performance Analysis Center):," in Business Process Management3rd International Conference, vol. 3649, France, BPM, pp. 109-119.

[22] R.B.Halima, K.Guennoun , K.Drira , M.Jmaiel, (2008) "Non-intrusive QoS Monitoring and Analysis for Self-Healing Web Services," in 1st IEEE International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2008), pp. 549-554.

[23] M. Tian, A. Gramm, H. Ritter, J. Schiller, (2004) "Efficient Selection and Monitoring of QoS-aware Web services with the WS-QoS Framework," in WI '04 Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence, pp. 152-158.

[24] F.Rosenberg, C. Platzer, S. Dustdar, (2006) "Bootstrapping Performance and Dependability Attributes ofWeb Services," in ICWS '06 Proceedings of the IEEE International Conference on Web Services, Chicago, pp. 205-212.

[25] W.John,S.Tafvelin, T.Olovsson, (2010) "Passive internet measurement: Overview and guidelines based on experiences," Computer Communications, vol. 33, no. 5, p. 533–550.

[26] J.Han,M.Kamber, (2006) Data mining :Concept and Techniques, 2nd ed. San Francisco, CA: Diane Cerra.

[27] J.K.Sharma, (2010) Fundamentals of Business Statistics. India: Dorling Kindersley.