QWS-Search: A Novel QoS Driven Web Service Discovery Framework

R.Jeberson Retna Raj [1], T.Sasipraba[2], Surya[3]
[#] Department of Information Technology, Faculty of Computing, Sathyabama University,
Jeppiar Nagar, Rajiv Gandhi Road, Chennai, India
[1] jebersonin@yahoo.co.in

Abstract— Web service discovery is an important process in services computing paradigm. Due to the enormous number of service providers available for a given functionality, identifying the suitable service to the client requirement is a challenge. The traditional discovery mechanism supports syntax-based discovery, and it limited support on semantic discovery. Web service registries and portals does not support client's QoS requirements. The general purpose search engine such as Google, Baidu may provide the required results for a search query. Since these search engines are mainly designed for retrieving web documents, they are well suited for discovering web pages and not web services. Traditional methods based on Term frequency (TF) and Inverse Document Frequency (IDF) limited support for web services discovery process. The repetition of terms and finding the richness of the document will not be applicable for finding appropriate web services. Due to the fact that WSDL contains much less information than web pages, these service discovery mechanisms are not applicable for discovering web services. To alleviate these difficulties, the Quality Driven Web Service discovery Framework QWS-Search has been introduced. The system consists of functional and non functional computations so that the returned results is semantically indexed and list them based on quality. The system QWS-Search has been tested with more than 2000 real world web services, and the result reports that the proposed system which outperforms the existing implementations in terms of precision and recall values.

Keyword-Web service discovery, QoS Normalization, web services, Quality web services.

I. INTRODUCTION

The emergence of web services has created unprecedented opportunities to numerous service providers for creating their internal business processes as web services and makes it accessible through web. Besides, clients used to locate the required services and access it via the internet. Due to the emergence of web services, many service providers are readily available to satisfy a client's request. Web service discovery is an important process in services computing paradigm. The traditional discovery methods in UDDI supports functional search that follows syntax based discovery. It has the limitation that it excludes semantic understanding of queries and support of non functional QoS requirements. More importantly, it may miss the most valuable services because of different meaning of services [2][11].

Existing QoS based selection architectures focus on nonfunctional requirements. In fact, it is a time consuming process as each time the QoS manager matches the client request to the local QoS database as well as UDDI registry. Generally, the traditional system for web service discovery gives more importance to the functional requirements. In services computing, it is fair that web services can be indexed based on performance. Furthermore, the services can be invoked either based on performance or cost. Some services are semantically relevant to the user query, but the usage of that service may not suit to the user's expectation. Therefore, merely selecting the services based on functional discovery is not a right way for satisfying the user needs. It is not fair as dealing with either functional or nonfunctional requirements in web services. We have to give equal importance to both of them so that the end user requirement is really fulfilled. The proposed QoS driven discovery focuses on finding the similar web services based on functionality and list them based on quality.

Existing web service discovery system like URBE (UDDI Registry By Example) [10] is based on semantic discovery. In that framework, the similarity measurement is based on considering the services with an equal number of terms which means semantic representation of a service. For example, web service S1 has two terms and service S2 has three terms, the system URBE considers only first two terms of S2 for similarity measurement. Rest of them are treated as left out terms, which are not considered for similarity measurement. The proposed system considers both equal and unequal number of terms for similarity measurement.

To address the aforementioned difficulties, the Quality driven discovery system has been introduced to assist the user to search the pertinent web service. In this approach, the bipartite graph algorithm is used to calculate the similarity between web services. To calculate the similarity measurement, the WSDL files are parsed and decomposed. The similarities between services are calculated using the semantic distance obtained through the Google search API. Existing systems consider each word or term in the WSDL as isolated. Since

each term in the WSDL is important, the proposed system considers every single term for finding the similarities. QoS values of web services are normalized, and the respective values are stored in the database. The calculated similarity measurement and QoS normalization values are utilized in displaying the web service search results to the users. Web service with higher rank value is displayed at the top of the search results. The proposed system has been tested with 2507 real world web services and the test result shows its accuracy with high precision and recall values.

A. Need of QoS driven discovery

The following scenario describes four web services W1, W2, W3 and W4, all being functionally similar web services. The web service name and operation name are partially similar for W3 and completely dissimilar for other three web services. But these services are functionally similar.

W1: Converter

Operation : GetCurrencyRate

Input : CurrencyFrom, CurrencyTo

Output : RateDate

W2: ExchangeRates

Operation : CurrentConvertToEUR

Input : dcmValue, strBank, strCurrency

Output : intRank

W3: CurrencyServer

Operation : CurrencyToCountry

Input : licenseKey, currency,

output : returnCountry

W4: iService.co.za_x0020_-_x0020_Finance

Operation : ConvertCurrency

Input : fromCurrency, toCurrency

Output : amountToConvert

If it is a syntax-based search, then for the query ‘currency’, it matches with only one service W3 and others are not retrieved. But, all the services are functionally similar. In the syntax-based discovery, the query is matched with the service name in the service description file. The underlying semantics may not be the same with the description. Therefore, text document discovery is not suitable for web services and semantic oriented discovery is the need of the hour.

B. Issues in QoS driven discovery

URBE (UDDI Registry By Example) has been proposed by Pierluigi Plebani et al.[10] for web service retrieval. The model considered port name and operation name similarity for tuning the performance. The semantic distance between terms from two compared services is employed by the lexical database WordNet. The similarity distance between the two web services is calculated by the bipartite graph algorithm. This model considers the two web services with an equal number of terms. However, the model does not give importance to the unmatched left out terms. Fangfang Liu et al. [5] proposed an approach for web service discovery. In this approach, the similarity between web services is computed based on WSDL. The model considered only the input and output parameter names for similarity measurement. Since the search interest of the clients often change, identifying the pertinent web service becomes challenging.

- Existing system return the results that are meeting the demand of client’s QoS soft constraints and not connected to functional hard constraints.
- Existing QoS discovery returns the results based on QoS and ordered only according to the QoS metrics.
- The web services whose functionalities are not exactly equivalent to the user search query are completely excluded from the result list.
- The existing text document search approaches are insufficient in the Web service environment, because Web services contain much more complex structure with very little text description.
- Existing system considers web services with an equal number of terms, and unequal number of terms will not be considered for similarity measurement.

This enormous hardship increases the complexity of discovering the appropriate web services. Furthermore, the existing service discovery mechanism returns a large number of web services, which may not be precise for

a query when the repository contains large service entries. The rationale is to propose a Quality driven web service discovery framework that assists the user for getting the pertinent web service.

C. Web service discovery based on ranking

The existing QoS supported registry only focuses on nonfunctional requirements and it lack the means of giving importance to functional requirements. It is not mandatory that the name of the web service and its operations should be alike. Each web service interface description has a different service name and has a list of operations, which will not reflect in the service name tag. Therefore, merely matching the service name with the user query based on syntax matching will not return better results. Existing approaches deeply concentrate on either functional or non functional requirements. However, approaches are giving less importance to both functional and nonfunctional requirements. The proposed approach gives equal importance to both functional and non functional requirements.

II. QoS DRIVEN WEB SERVICE DISCOVERY ARCHITECTURE

The proposed QoS driven web service discovery consist of functional and non functional computations.

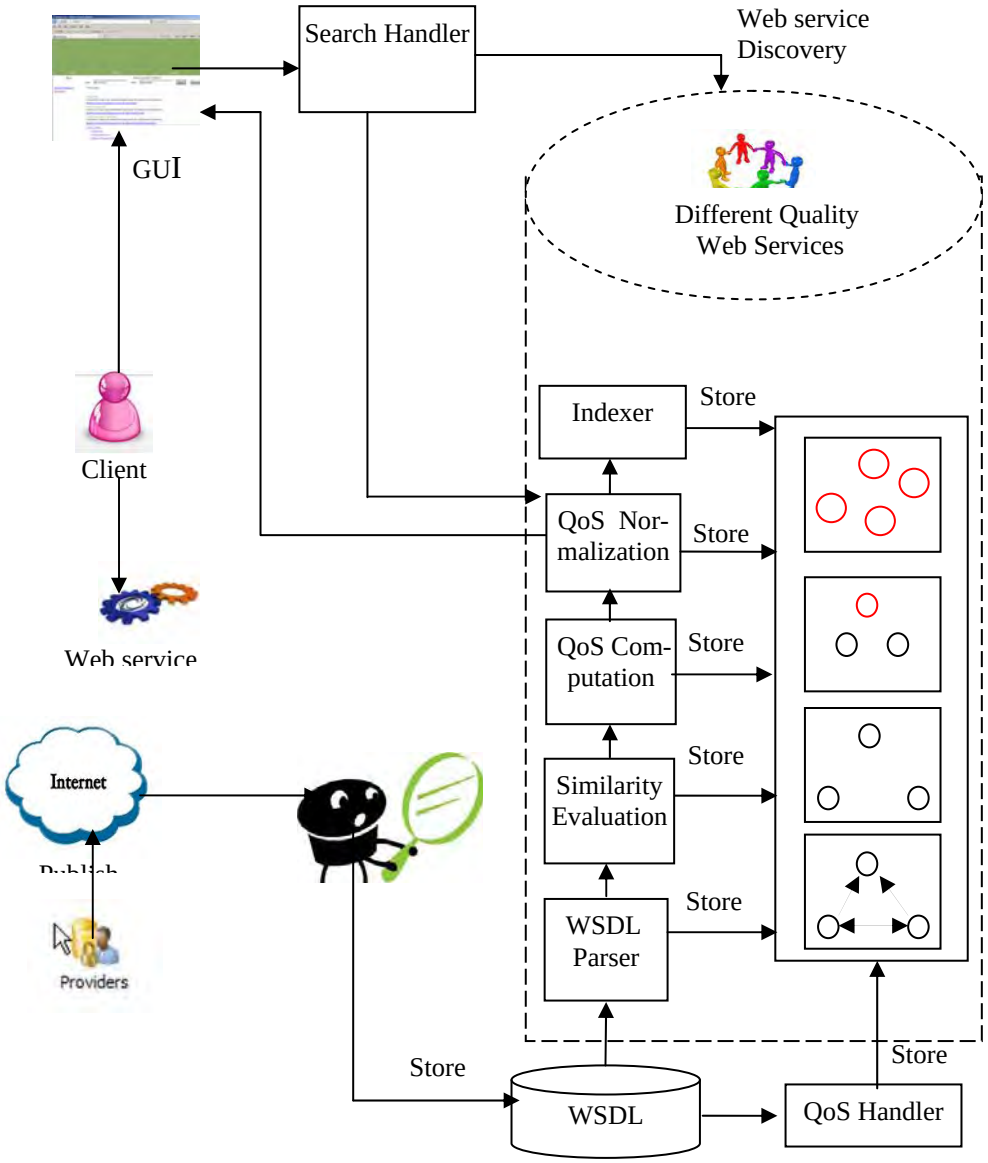

Fig.1. Proposed QoS driven web service discovery architecture

The architecture of the proposed QoS driven web service discovery is shown in Figure 3.1. The overall system consists of offline and online processes. In the figure, the components belonging to the dotted region requires offline computation. The web service descriptions are retrieved from various web service portals and directories, store the WSDL files in the database. The WSDL parser extracts the Meta data from a WSDL file and stores it in the database. The extracted metadata contents such as Message, Port type, operation,

input/output is stored in the database. The QoS handler extracts the QoS attribute values and stores it in the database. The normalization algorithm is used to normalize the different QoS values and transform it in the range between [0, 1]. The similarity evaluation component is used to find the similarities between the two web services using a bipartite graph algorithm. The QoS values are computed by the enhanced prediction model. The indexer will index the web services based on the aggregated score of functional and non-functional values. These procedures will be done in the offline mode. The QWS Dataset [3] consisting of 2507 web services WSDL files are considered for validating the proposed system.

The gist of the proposed framework extends from the work done by Fangfang Liu et al. [5] for web service discovery. The system calculates the similarity between the web services based on WSDL file. The WSDL description consists of ports, operations, input/output messages and other definitions to express its function. WSDL acts as an interface between the service provider and consumer. Since WSDL reveals the exact functionality of the service, the proposed system considers it as the input source. Fig.2 shows the similarity measurement between compared web services.

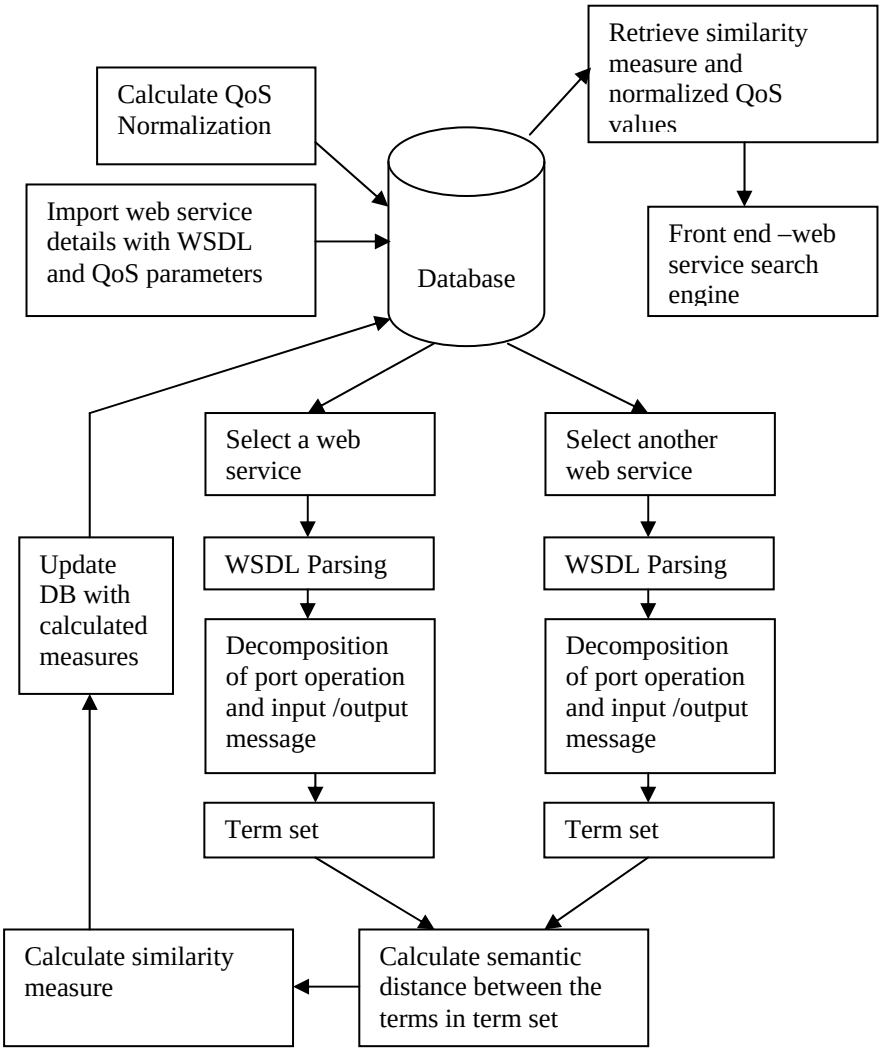

Fig.2. Similarity measurements of compared web services

The above diagram shows how the two services have undergone various step by step processes for similarity computation.

In this approach, the similarity between web services can be computed based on WSDL. The WSDL file is parsed, and the metadata elements such as port, operation, and input/output messages are extracted from a WSDL file for similarity measurement. Generally, the names of those extracted terms are a concatenation of words. Those concatenated words are decomposed using a decomposition algorithm. Soon after decomposition process over, a term set is obtained. The computed similarity measures are used in service classification and service query.

Considering two compared services as a bipartite graph includes an important assumption that there is no edge between terms within one term set. That's to say the terms that are originally utilized as a whole to represent the capability of a service are now only individuals. The implied connections are broken. Especially when two services have different number of terms, they have left out terms, which are out of consideration. Nevertheless, these left out terms also have an effect on the similarity of services. Considering the comparison of services in detail, when the connections between terms are broken, it's much possible that the real semantics of the terms are no longer kept. The "unmatched" terms definitely have an effect on the similarity of services.

Similarity measurement of two web services is calculated from the term set obtained after decomposition of each web service. Term set of one web service is compared against a term set of another web service. Comparison is done by calculating the semantic distance between each and every term in one term set with terms in another term set. This approach uses new metrics to deal with the similarity of services which employ the terms inside services fully and thus can reflect the association between the terms.

Using the Google search API, semantic distance between two terms is obtained. The Google search API returns the number of web results containing the given term. Once the semantic distance is calculated between the terms in the term set, higher distance will be selected, and similarity measurement is calculated.

The proposed system considers the web service QoS parameters such as Response time, Availability, Throughput and Reliability for nonfunctional computations. These QoS parameters determine the quality of the web services. The QoS values are normalized for every web service. The calculated QoS normalization values determine the ranking of the web services listed in the search results. Web services having high normalization values are listed at the top of the search results. The following functional and non-functional computations are required for measuring the similarity between web services.

- Crawling of web services
- WSDL Parsing
- Decomposition of parsed WSDL
- Semantic distance computation
- Similarity measurement
- QoS Normalization and indexing

A. Crawling of web services

The WSDL files are retrieved from various directories and portals from the Internet. Web service details are obtained from the web through RSS feeds provided by the websites. Details like web service name, WSDL file path and optional QoS parameters like Response time, Availability, Throughput, Successability, Reliability, Compliance, Best practices, latency and documentation. QoS parameters determine the quality of the web services available. With the help of CURL functions in PHP (Personal Home Page), the WSDL file contents are obtained from the WSDL file path retrieved from Rich Site Summary (RSS) feeds. The downloaded WSDL file content is considered as the input for further operations.

B WSDL Parsing

The WSDL-parser parses the metadata contents such as message, type, operations, and service name from the WSDL description of a web service, and gets stored in the database. The WSDL file is an XML file which contains all the required information related to operations performed by the web service. It is an important step where the port name, operation name, input/output messages are retrieved using XML parsing techniques available in PHP. Fig.3. shows the excerpts of a sample WSDL file.

```
<wsdl:portType name="DispatcherAPI">
<wsdl:operationname="getURL" parameterOrder="orgCode">
<wsdl:input message="impl:getURLRequest" name="getURLRequest"/>
<wsdl:output message="impl:getURLResponse"  name="getURLResponse" />
</wsdl:operation>
 </wsdl:portType>
```

Fig.3 Excerpts of a WSDL file

The following details are retrieved after WSDL parsing,

Port name = DispatcherAPI

Operation name = getURL

Input message = impl: getURLRequest

Output message = impl: getURLResponse

C. Decomposition of parsed WSDL

Once the WSDL parsing is done, the retrieved port, operation, input/output message names are decomposed for further processing. Decomposition is done based on the following rules,

Upper case letters are converted to lower case.

Hyphens are being converted to comma.

Numeric characters are removed.

Reference for the rule is available in the following Table I,

TABLE I
Decomposing rules for wsdl

Rule	Name	Word
Case change	getURL	get, url
Suffix number elimination	film1	film
Underscore separator	from_currency	from, currency

Decomposition algorithm

The aim of the decomposition algorithm is to split the names of ports, operations and input/output messages available in WSDL. For an input of any WSDL file, the output term set is generated. The following steps are required for decomposition.

Step1: Read the WSDL file content.

Step2: Perform XML parsing.

Step3: Retrieve the port, operation, input and output messages.

Step 4: Splits the retrieved port name with uppercase letters.

Step 5: Convert all the available upper case letters to lower case.

Step 6: Remove the Hyphens available in port names.

Step 7: Remove the numerical characters available in the port names.

Step 8: Repeat Step 3 to Step8 for operation, input and output messages.

Application of decomposition algorithm

Before Decomposition,

Port name = DispatcherAPI

Operation name = getURL

Input message = impl:getURLRequest

Output message = impl:getURLResponse

After Decomposition,

Decomposed port name = dispatcher, api

Operation name = get, url

Input message = get, url, request

Output message = get, url, response

Once the decomposition is done, the decomposed port, operation, input and output message terms are inserted into the database.

D. Semantic distance computation

Semantic distance computation is required for finding the similarity between the two web services. Two web services already decomposed are considered for semantic distance computation. Number of terms available in term set of two web services can vary. In this system, each and every term available in the term set is considered important. Though, the number of terms mismatch, semantic distance between the mismatch terms with other terms is calculated. Semantic distance between the terms available in two different term sets are obtained using Google search API. The Google search API returns the number of web results obtained for a term. Semantic distance between the two terms are obtained using the following formula [5],

$$term\_dis\tan ce_{1,2} = \frac{\log\left(\frac{N(k_1 \cap k_2)\times N}{N(k_1)\times N(k_2)}\right)}{\log N} \quad (1)$$

where N is the Number of total web pages and usually set to 10^{11}. k_1 is a term available in term set 1. k_2 is a term available in term set 2. $N(k_1 \cap k_2)$, $N(k_1)$, $N(k_2)$ are returned by the Google search API. $N(k_1 \cap k_2)$ denotes the co-occurrence of terms k_1 and k_2 in Google search results. $N(k_1)$ denotes the occurrence of the term k_1 in the web pages from Google. $N(k_2)$ denotes the occurrence of the term k_2 in the web pages from Google. Semantic distance between the two terms can also be obtained using the following Google normalized distance formula,

$$NGD(x,y) = \frac{\max\{\log f(x), \log f(y)\} - \log f(x,y)}{\log M - \min\{\log f(x), \log f(y)\}} \quad (2)$$

where x is a term available in term set 1 and y is a term available in term set 2. In this formula, the Normalized Google Distance $NGD(x,y)$ represents the Google semantic distance and M represents the total web pages that Google search engine obtains. $f(x)$ represents the return pages of Google search engine containing term x. $f(y)$ represents the return pages of Google search engine containing term y. $f(x,y)$ represents the return pages of Google search engine containing term x and y.

E. Bipartite graph construction

The graphical representation of extracted terms for the compared web services are shown in Figure 3.5. The similarity of services is based on maximum weight matching of bipartite graph. For a given graph G= (S1, S2, E), where S1 and S2 be the two web services and S1={ki}and S2={kj}. ki and kj are the terms in their corresponding WSDLs. $E = (< k_i, k_j >, W_{i,j})$, where $W_{i,j}$ is the semantic distanced between terms k_i and k_j. If G is a bipartite graph then S1 and S2 are two disjoint subsets such that there exist no edges between the vertexes within the same set. In the maximum weight matching $M \subseteq E$ of G, no two edges share the common end vertex and sum of the weights of M is the maximum [5]. The maximum weight match M of G is:

$$\max\_value = \max\{\sum_{k_i \in S_1}^{k_j \in S_2} W_{i,j}\} \quad (3)$$

where $0 \le W_{i,j} \le 1$

When two services have the same cardinality, i.e. $|S_1| = |S_2|$, the graph is a balanced graph. For a balance graph, the terms in S1 and S2 are connected which is shown in Fig.4. Otherwise, for an unbalanced graph there exist a unmatched term as in Fig.5.

Once the semantic distance between two different term sets are calculated using the Google search API, a bipartite graph is constructed to show the calculated semantic distance between the terms. Fig.4 shows the bipartite graph for equal number of terms. To find the semantic distance between two terms k_{11} and k_{21}, first k_{11} is sent to Google search API and number of web pages containing term k_{11} are returned as output. Google search API then returns the number of web pages containing the term k_{21}. Then both terms are given to the Google search API to calculate the total number of web pages containing both the terms. Once the number of web pages containing term1, term2 and term1 & term2 is calculated, semantic distance between k_{11} and k_{21} can be obtained using the above formulas. Then the semantic distance between k_{11} and k_{22} are calculated. The same process repeats for terms k_{12} & k_{21} and terms k_{12} & k_{22} are calculated. The maximum semantic distance between the terms are used to calculate the similarity measure.

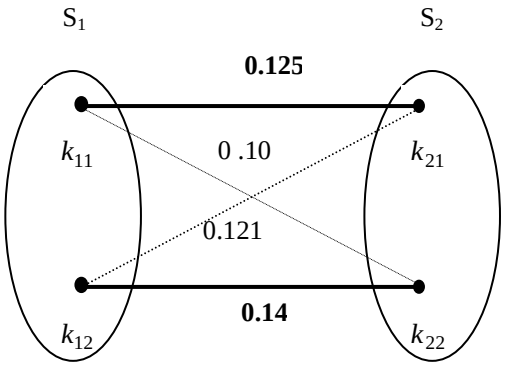

Fig.4. Bipartite graphs with equal terms

The maximum weight matching of two services in Fig.5 is 0.125 + 0.14 = 0.265.

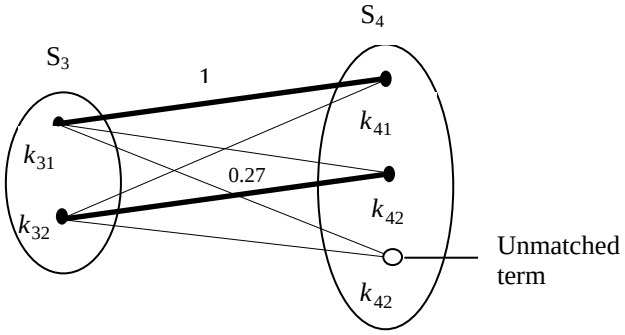

Fig.5 shows the bipartite graph for unequal number of terms.

F. Semantic distance calculation algorithm

The objective of the semantic distance calculation algorithm is used to find the semantic connection between the terms using Google web search results. The term sets of two web services are given as a input and the algorithm returns the semantic distance between the terms available in different term sets.

Step 1:Using Google search API, calculate the number of web results returned for a term x available in one term set.

Step 2:Obtain a term y available in another term set and calculate the number of web results returned using Google search API.

Step 3: Calculate the number of web results returned for both the terms x and y .

Step 4: Calculate semantic distance between the terms using any one of the formulas (1) and (2).

Step 5: Repeat the step 1 to 3 to all available terms in both term sets.

Step 6: Find the maximum semantic distance between the terms.

G. Similarity Measurement

Similarity measure for compared web services of equal number of terms and unequal number of terms can be calculated. Highest semantic distance between the terms is retrieved and similarity measure is calculated using two different formulas (4) and (5). The similarity degree of web service S1 and S2 is the normalized factor of equal and unequal number of terms in the respective term set. Similarity measure for equal terms can be defined as [5],

$$Similarity_{1,2} = \frac{\max\_value_{1,2}}{|M|} \tag{4}$$

where $0 \le Similarity_{1,2} \le 1$ and $|M|$ is the number of edges.

Similarity measure for unequal terms can be defined as,

$$Similarity_{1,2} = \frac{2\max\_value_{1,2} + \sum_{k_j \epsilon U\_MV} \max\{w_{i,j}\}}{|S_1| + |S_2|} \quad (5)$$

where $0 \le Similarity_{1,2} \le 1$ and $|S_1|$ is the number of terms available in term set of web service 1. $|S_2|$ is the number of terms available in term set of web service 2. U_MV is the unmatched term and $\max\{w_{i,j}\}$ is maximum semantic distance between the left out term with other terms.

Similarity measurement algorithm

The aim of the similarity measurement algorithm is to find the similarity between two web services. The maximum semantic distance between the terms are taken as input and the algorithm returns the similarity measure between two web services.

Steps:

Check whether the similarity measure is to be calculated for equal terms or unequal terms.

If equal terms then apply the formula (4).

If unequal terms then apply the formula (5).

H. QoS normalization

The QoS normalization and ranking is required for transforming the vector of multidimensional QoS value into single metric. Assume that all services are registered with their respective parameters. QoS parameters of the web services determine the quality of the web service. In this work, QoS normalization is done by using the following formula [11]:

$$Utility_i(q_i) = \begin{cases} \frac{q_i - (q_i)_{\min}}{(q_i)_{\max} - (q_i)_{\min}} & if Q_i \in QoS^{Positive} \\ \frac{(q_i)_{\max} - q_i}{(q_i)_{\max} - (q_i)_{\min}} & if Q_i \in QoS^{Negative} \\ 1 & if\ (q_i)_{\max} - (q_i)_{\min} \end{cases} \quad (6)$$

$QoS^{Positive}$ denotes the positive QoS attributes which represents Throughput, Reliability and Availability. These attributes based on higher-is-better policy can be followed. If the values of availability and reliability are higher, then the normalized values are higher. $QoS^{Negative}$ denotes the negative attributes and response time is come under the category. These attributes are based on 'lower-is-better' principle. If the response time is lower, then the normalized value is higher.

q_i is a QoS parameter value of a particular web service. $(q_i)_{\min}$ is minimum QoS parameter value of all available web services. $(q_i)_{\max}$ is maximum QoS parameter value of all available web services. In the front end, calculated QoS normalization values are used to rank the web service search results. Web service with higher QoS values are listed on the top.

III. AGGREGATING FUNCTIONAL AND NONFUNCTIONAL SCORE

The aim of the proposed QoS driven discovery is to focus on both functional and non-functional requirements. For that, the functional score and non functional score can be aggregated. The semantic distance of underlying terms such as portType, message and input/output of compared web services are added with QoS distances of the compared web services.

In our earlier implementation, the functionally similar services are identified based on semantic comparison and ranked based on quality. The system identifies the similar services which are grouped together functionally. According to the QoS score the services are listed to the user query. This kind of approach is again giving more importance to non-functional information and less to functional requirements. Therefore, the proposed approach computes the similarity measurement for both functional and non-functional information of compared web services.

The functional similarity between compared services in terms of normalized factor is added with the normalized score of non-functional QoS values. In this work, the similarity measure between two web services Si and Sj can be computed by using the proposed formula,

$$Sim(S_i, S_j) = \sum_{i \in S} \frac{f(S_{i,j}(d_i))}{N} + \left[d(S_i \left| \sum_{i \in S} \frac{nf(q_i)}{N} \right|) - S_j(\left| \sum_{i \in S} \frac{nf(q_i)}{N} \right|) \right] \quad (7)$$

where $Sim(S_i, S_j)$ is the similarity between two web services Si and Sj, d_i is semantic distance between parametric representations of portType, operation, input and output parameters of a web service. N is the total number of WSDL elements considered for similarity measurement. The term $nf(q_i)$ is the non-functional normalized score value of web service Si and $dS_{i,j}$ is the non-functional distance between web services Si and Sj.

A. Web Service Search

Calculated similarity measure and QoS normalization values are used in web service search. Front end contains the web service search. User can search for their required web services through three options. The options are 'search by keyword', 'search by WSDL' and 'search by input/output messages'. When the user searches by keyword, only the web services whose name and description matching with the given query is listed.

When the user searches by WSDL, the calculated similarity measure and QoS normalization values in the back end are applied in displaying the search results. The search results are listed based on the QoS normalization values. The web service having higher QoS normalization value is listed at the top. The web services which are matched with the given query is listed along with similar web services whose similarity measure value is greater than or equal to the threshold value.

When the user searches by input/output operations, the calculated similarity measure and QoS normalization values are applied in displaying the search results. The web service whose input or output messages matching with the given query is listed along with its similar web services whose similarity measure value is greater than or equal to the threshold value. Threshold value used is 0.5. Once the search operation is done, database table 'search term' is updated with user's given query, search mode and the total number of results obtained.

B. QWS Dataset

The proposed system using QWS data set consist of 2507 web services with QoS attributes like Response Time, Availability, Throughput, Successability, Reliability, Compliance, Best Practices, Latency, Documentation, Service Name, WSDL Address[2]. The proposed system considers Response time, Throughput, Reliability and Availability as required parameters for the discovery process. The reason for not considering the other attributes is discussed in Chapter 1.

Sample QoS data for weather web service

103,85,16.1,95,73,78,80,0.89,91,DOTSFastWeather,http://ws2.serviceobjects.net/fw/fastweather.asmx?wsdl

261,100,1.8,71,58.1,78,80,229.5,94,71,2,WeatherForecast,http://www.webservicex.net/WeatherForecast.asmx?wsdl

160,100,2.2,71,73.3,78,84,74,32,71,2,WeatherFetcher,http://glkev.webs.innerhost.com/glkev_ws/WeatherFetcher.asmx?wsdl

3356.5,91,1.5,97,60,78,79,19.07,91,Service,http://ejse.com/WeatherService/Service.asmx?wsdl

285,85,4.2,95,73,78,84,96,38,GlobalWeather,http://www.webservicex.com/globalweather.asmx?WSDL

642.5,72,5.4,72,67,89,72,25,41,ndfdXML,http://weather.gov/forecasts/xml/SOAP_server/ndfdXMLserver.php?wsdl

409.33,49,1.8,27,41.4,89,72,401.5,96,55,4,ndfdXML,http://www.weather.gov/mdl/XML/Ccode_test/DWMLgen/wsdl/ndfdXML.wsdl

IV. EXPERIMENTAL RESULTS

The proposed system considers each and every term of the message, portType, input and output messages during similarity measurement. The main purpose of similarity measurement is to list the related web services to the consumers, so that they can have more options to select their required web service. To validate the proposed system, more than 2500 real-world web services are considered. QWS dataset (Eyhab Al-Masri et al. 2008) has been used for nonfunctional computations [3].

The QoS values are measured by using enhanced WSRec [9]. When the number of operations is equal in web service S1 and S2, the similarity measurement between two operations is measured by equation (3). If the number of operations is unequal, the similarity between web services is calculated by equation (4).

QoS normalization values are calculated for each and every web service. Calculated QoS normalization values determine the ranking of the web services listed in the search results. Web services having high

normalization values are listed in the top of the search results. Each and every web service is decomposed, and so each web service contains more than one port, operations and input/output messages.

For example, consider two web services, let S1 be “phoneService” and S2 be “EmailValidation.” The detailed process of WSDL parsing, decomposition, semantic distance computation, similarity computation, QoS normalization, and similarity measurement are illustrated with that example.

WSDL parsing

In the WSDL parsing the metadata of WSDL such as Port type name, operation name, input and output messages are extracted. For the given example, the following details are obtained from the WSDL of web service S1 by parsing,

Port type name: phonePhonePort

Operation name: getSecret

Input message: getSecret

Output message: getSecretResponse

Similarly, the following details are obtained from the second web service S2 WSDL by parsing,

Port type name: xWebEmailValidationInterface

Operation name: validateEmail

Input message: validateEmailIn

Output message: validateEmailOut

Decomposition

The importance of decomposition is to preprocess the parsed contents. The parsed details are decomposed using the decomposition algorithm and the terms are obtained. Web service S1 details are decomposed and the resultant terms are as follows,

Port type name : phone, phone, port

Operation name: get, secret

Input message : get, secret

Output message: get, secret, response

Similarly, web service S2 details are decomposed and the resultant terms are as follows,

Port type name : x, web, email, validation, interface

Operation name : validate, email

Input message : validate, email, in

Output message : validate, email, out

Semantic distance calculation

Once the details are decomposed, the terms are obtained. The semantic distance between the terms are calculated and represented using bipartite graph. The semantic distance between the terms of port, operation, input and output messages are calculated and represented using bipartite graph. Table II shows the semantic distance between two web service port names similarity.

TABLE II
Semantic distance between ports of S1 and S2

Terms	**Semantic distance**
x, phone	0.056
x, port	0.007
web, phone	0.061
web, port	0.052
email, phone	0.062
email, port	0.070
validation, phone	0.068
validation, port	0.093
interface, phone	0.083
interface, port	0.109

For example, the semantic distance between the terms x and phone is obtained from Google and the value is 0.056. Similarly, the semantic distances between other terms are obtained. According to the bipartite graph algorithm, the maximum weight matching term is considered. Fig. 6 shows the bipartite graph constructed for port similarity between web services s1 and s2.

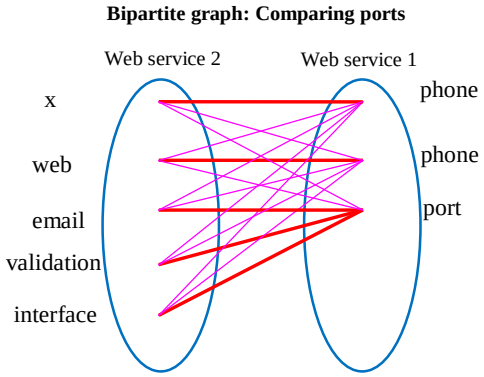

Fig.6 Bipartite graph shows semantic distance between ports of S1 and S2

Fig.6 shows the bipartite graph for similarity between two web services S1 and S2 with port name similarity. Here, the semantic distance between the terms 'x' and 'phone' is 0.056, and 'x' and port is 0.007. According to the graph showing in Fig.6, the maximum weight matching terms are connected for similarity measurement. Therefore, the maximum weight matching terms 'x' and 'phone' is connected. Similarly, other terms are mapped. Similarly the semantic distance of terms for operation, input and output are computed.

For similarity measurement, the maximum weight matching terms between compared web services are considered. The maximum weight matching terms in port element is added together to obtain the portType similarity between compared services.

TABLE III
Similarity measure between web service S1 and S2

Similarity Measure	Port	Operation	Input message	Output message	Functional Score
S1 & S2	0.326	0.048	0.108	0.066	0.137

Table III shows the Port, operation, input and output similarities between web service S1 and S2. For example, the port similarity between web services S1 and S2 is 0.326.

QOS normalization for service S1 and S2

The QoS normalization is used to normalize different QoS values and transform it to [0, 1]. The QoS of web services such as response time, throughput, reliability and availability are normalized using the equation (6).

TABLE IV
QoS normalized values of S1 and S2

Service	Response time	Availability	Throughput	Reliability	score
S1	0.088	0.818	0.02	0.607	0.383
S2	0.024	0.807	0.517	0.893	0.560

Table IV shows the normalized QoS values of web service S1 and S2.The nonfunctional score of web service S1 is obtained through taking average of normalized QoS values of response time, availability, throughput and reliability. Similarly, the nonfunctional score of service S2 is calculated. Table 3.7 shows the nonfunctional score of service S1 and S2. The difference between the nonfunctional score S1 and S2 is calculated.

Similarity between web service S1 and S2

The similarity between web service S1 and S2 is calculated by aggregating the functional score and the nonfunctional score values. Table V shows the similarity between web service S1 and S2 based on their functional and nonfunctional scores. Similarly, the similarity is calculated for other comparable services and indexed.

TABLE V
Similarity between S1 and S2

Web Service	Functional score	Non functional score	Total score
S1 and S2	0.137	0.177	0.314

Table V shows the total score value consisting of functional and nonfunctional computations. The similarity between web service S1 and S2 is 0.359 which is based on functional and nonfunctional similarity measurements. Here, the functional similarity between web service S1 and S2 is 0.137 which is less than the threshold value 0.5, so that they are not considered as similar services. In this way, other services are compared and indexed.

A. Performance Analysis

The system has been implemented with the support of following system configurations. The offline computation and online processing of queries can be implemented with the support of PHP 5.3.0 server side scripting language. Apache 2.2 is a web server for client server interaction. MySQL 5.1 acts as a database server for backend. The hardware configuration for the processor of Intel Pentium IV, speed of RAM will be 2GB with the hard disk capacity of 40 GB.

The approach used in this work provides greater precision and recall values, since it depends upon the semantics of the terms available in the WSDL. Google search API is used to find the semantic distance between the terms. The system is tested with three web service categories such as currency converter, weather and address validation services. The precision and recall values are computed from the following.

$$precision = \frac{|\{relevant\ items\} \cap \{retrieved\ items\}|}{|\{retrieved\ items\}|} \tag{8}$$

$$recall = \frac{|\{relevant\ items\} \cap \{retrieved\ items\}|}{|\{relevant\ items\}|} \tag{9}$$

TABLE VI
Performance measures of three clusters

Cluster	Proposed framework		Existing approach Khalid (2010)	
	Precision%	Recall%	Precision%	Recall%
Currency exchange	89.4	94.7	84.2	88.9
Weather	87.5	100	70.0	87.5
Address validation	81.2	93.7	60	93.7

Table VI shows the performance analysis for functional similarity of the proposed approach with the existing approach. The proposed approach provides higher precision and recall values due to the introduction of robust similarity measurement algorithm. The existing model employs the lexical database 'WordNet' for semantic distance calculation. Due to the fact that some of the terms are not supported by 'WordNet', the existing system is not precise enough for similarity measurement. For currency exchange web service, the precision and recall is 87.5% and 94% based on the proposed framework as against 84.2% and 88.9% respectively. For the other categories 'weather' and 'address validation', the proposed system achieves more than 10% in terms of precision and recall values comparing to the existing approach. The test results show that the proposed approach provides better performance than the existing approach in terms of high precision and recall values.

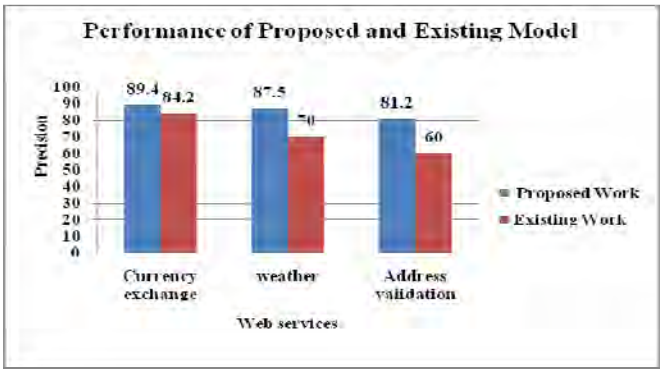

Fig.7 Comparison of precision value for existing and proposed method

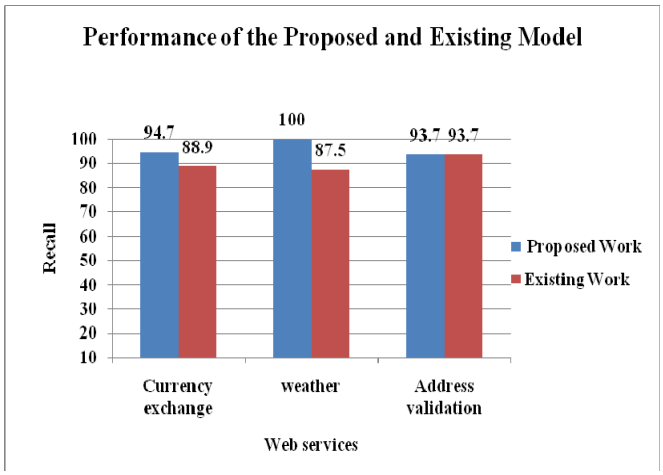

Fig.8 Comparison of recall value for existing and proposed approach

Fig.7 and 8 shows the comparison of precision and recall between existing and the proposed method. The Address validation service cluster, the recall value is same as 93.7% with the existing approach. The service WSDL document http://142.176.62.103/ GEONOVA_WS/CivicAddressPointRange.asmx?WSDL defines addresses in a different manner than the other Web services and it uses a large number of acronyms. Therefore, the proposed approach is not able to group it correctly.

TABLE VII
Functional support of proposed and existing approaches

Models	Similarity Measurement	Keyword	Keyword, Input/Output	QoS Support
Woogle(2004)	TF/IDF	no	yes	no
WSExpress(2010)	TF/IDF	yes	yes	yes
CoWS(2011)	TF/IDF	yes	No	yes
URBE(2009)	Bipartite graph	no	yes	no
Homogeneous web service discovery (2009)	Agglomerative	no	yes	no
Proposed QoS driven discovery	Bipartite graph	yes	yes	yes

Table VII shows various existing models and their functional support for web service retrieval. Comparing with these models, the WSExpress provides QoS support which is based on TF/IDF. In CoWS, the reputation is the QoS attribute used for nonfunctional computation and functional computation based on TF/IDF. Since the web service description has little information, finding the repetition of terms and applying the TF/IDF is not enough for similarity measurement. In URBE, the compared web services with equal number of terms are considered for similarity measurement. Since the left out terms are also important for similarity computation, the returned results of URBE are not precise. The proposed approach gives importance to the left out terms for similarity measurement by making use of Google search results for semantic distance calculation. Furthermore, the proposed model precisely returns the web services for a user query based on the importance of both functional and nonfunctional weights. Therefore, the Quality Driven Discovery system is good enough to assist the user in locating the desired web services.

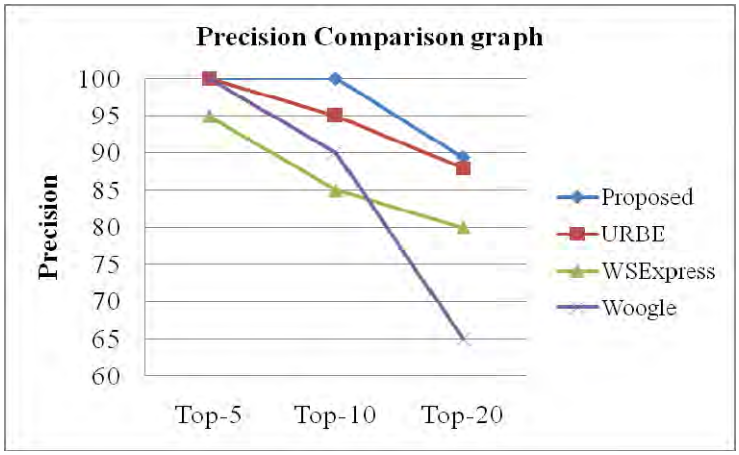

Fig.9 Precision comparisons for Top-k results

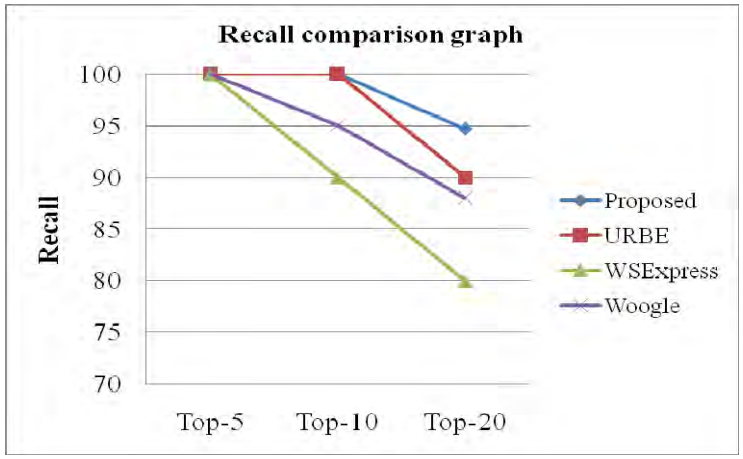

Fig. 10. Recall comparison for Top-k results

Fig.9 and 10 shows the precision and recall comparison for top-k results for existing and proposed model. The precision for top-5, top-10 and top-20 results are analyzed. The proposed model outperforms the existing implementations. The precision of the proposed system for the top-5 and top-10 are higher, which means the proposed system returns the very relevant services at the top of the results. Similarly, the recall value for the proposed method is higher comparing to the existing models. The similarity measurement in URBE is focused on the input and output messages of compared web services for increasing the precision and recall. The other details such as operation and port similarity are not considered. The proposed system considered port, operation, input and output messages for similarity measurement. Fig.11 shows the search results for the proposed discovery and the conventional syntax-based discovery.

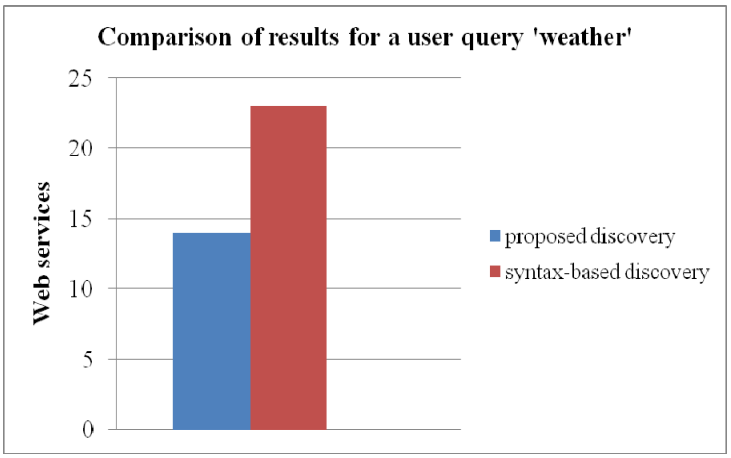

Fig.11 Search results for the proposed discovery and the syntax-based discovery

V. CONCLUSION

The conventional syntax-based discovery matches the user query with the available service description in the registry or repository. The proposed system calculates the similarity between the compared web services based on the semantic distance for operations, ports, input and output messages using Google search API. The search results are based on computation of functional similarity of compared web services and QoS ranking. For the user query 'weather', the syntax-based discovery returns all the services matched with the description. Totally twenty three services matches with the user query and returns to the user. Web service consumers are provided with a list of related web services, from which they can retrieve the pertinent web service. Due to the same query for the proposed discovery, the search results are reduced to fourteen. The system returns the semantically matched services for a user query.

REFERENCES

[1] Athman Bouguettaya, Qi Yu, Xumin Liu and Zaki Malik (2011), "Service-Centric Framework for a Digital Government Application", IEEE Transactions on Services Computing, January-March, Vol. 4, No. 1, pp. 3-15.

[2] Brian Blake M. (2009), "Knowledge Discovery in Services", IEEE Internet Computing, pp. 88-91

[3] Eyhab Al-Masri and Qusay H. Mahmoud (2008), "Discovering Web Services in Search Engines", IEEE Journal on Internet Computing, May/June, pp. 74-77.

[4] Eyhab Al-Masri and Qusay H. Mahmoud (2008), "Toward Quality-Driven Web Service Discovery", IEEE Journals & Magazines on IT Professional, May/June, pp.24-28.

[5] Fangfang Liu, Yuliang Shi, Jie Yu, Tianhong Wang and Jingzhe Wu (2010) "Measuring Similarity of Web Services Based on WSDL", IEEE International Conference on Web Services, pp.155-162.

[6] Khalid Elgazzar, Ahmed E. Hassan and Patrick Martin (2010), "Clustering WSDL Documents to Bootstrap the Discovery of Web Services", IEEE International Conference on Web Services, pp. 147-154.

[7] Kyriakos Kritikos and Dimitris Plexousakis (2009), "Requirements for QoS-Based Web Service Description and Discovery", IEEE Transactions on Services Computing, October-December, Vol. 2, No. 4, pp.320-337

[8] Richi Nayak Bryan Lee (2007), "Web Service Discovery with additional Semantics and Clustering: Queensland University of Technology", IEEE/WIC/ACM International Conference on Web Intelligence, pp. 555-558.

[9] Zheng Z.B., Ma H., Lyu M.R. and King I. (2009), "WSRec: a collaborative filtering based Web service recommendation system," Proceedings of 7th International Conference on Web Services (ICWS), pp. 437-444.

[10] Pierluigi Plebani and Barbara Pernici (2009), "URBE: Web Service Retrieval Based on Similarity Evaluation", IEEE Transactions on Knowledge and Data Engineering, November, Vol. 21, No. 11, pp.1629-1642.

[11] Jeberson Retna Raj and Sasipraba T "Web Service Recommendation Framework Using QoS Based Discovery and Ranking Process", IEEE – Third International Conference on Advanced Computing, ICoAC2011, MIT campus, Anna University, Chennai, India, ISBN : 978-1-4673-0669-0, pp.371-377,