Service Crawling using Google Custom Search API

Maria Allauddin
College of Electrical and Mechanical Engineering
National University of Sciences and Technology
(NUST) Islamabad, Pakistan

Farooque Azam
College of Electrical and Mechanical Engineering
National University of Sciences and Technology
(NUST) Islamabad, Pakistan

ABSTRACT
The area of Web Service Discovery (WSD) is a primary area of research today. It has root importance for utilizing web services for personal or organizational needs. However the users of web service are yet facing a challenge to find the desired web service due to rapid growth of web services available on internet. There is a need of a strategy to locate web services with issues covering like performance, flexibility and reliability across multiple heterogeneous registries, which is a challenging task yet. Our proposed framework covers the limitations of current techniques; it actively obtains user required web service by crawling among different repositories. We have used Google Custom Search API for this purpose. The search is both interface based and functional level and there is flexibility to add more links to expand the needs of user request. We have performed some verification and validation checks to confirm the retrieved document is a web service and is currently available.

General Terms
Web Service Discovery

Keywords
Web Service, Discovery, interface, functional, Google API.

1. INTRODUCTION
A wonderful trend in technology of the age is that Web services serve on internet as replacement of applications. Services are small components present on internet that cooperatively make a complete application environment.

Services have many characteristics that make them able to be a part of an architecture that is mutually service oriented, but it is also quality of service that it can function completely independently. So we can say that each service is accountable for its own operation as a whole. Due to independence of individual operation or service domain, the structure they have and their programming logic

need not obey to any specific platform or technology.

Web Services are applications that can be published to be found on internet and then invoked to give result of the operation defined in it.

A Web service can be an application component like: currency conversion, weather reports, or even dictionary as service. They also solve interoperability problems by providing a way to exchange data between different applications with different platforms. So they are gradually attaining preference as a technology among developers and businesses.

Services are registered in registries bye their providers. Usually in UDDI. For communication between providers and consumers SOAP messages are exchanged. HTTP protocol is used for such communications. It is becoming critical day by day to find the required service due to rapidly increasing number of web services available on internet. A procedure called Web Service Discovery is used to find the required needs.

UDDI search allows only specific keywords to be searched for example Service Name, key or category. So this was only interface based search.

Using different crawling open source applications the user is limited to search single domain at a time.

Google WSDL API was specific for wsdl search in web service discovery. For some reasons it has been depreciated.

We have used Google Custom Search API to provide the user search its required service. Though the API is not specific for web service search but we have customized it to produce results that are only web services.

Current approaches for service discovery have some limitations these are:-

1. Querying Heterogeneous registries at a time.

2. Retrieving up to date information on user's request.

3. In case of searching from web there is a need of in time response.

4. One time consuming task is that the users have to search whole registry each time they need a service. It requires a lot of effort.

5. Majority of current approaches, lack a reliable, stable and trust-worthy discovery.

6. Services are themselves heterogeneous i.e. they have different formats for exchanging data.

7. The web services published are tagged with a lot of information that makes a program difficult to trace out the required web service on given attributes.[2]

Keywords are used to discover web services in UDDI. Ranking services and filtering them is main advantage of UDDI. Main drawback is that search can only be made on basis of metadata so it limits the search criteria.

A. Contribution

We have proposed a framework to overcome some web service discovery problems. Using Google Custom Search API provides the flexibility to search the user query on more than one heterogeneous registry at a time. We have programmed to retrieve only relevant wsdl files that are valid and available. The

limitation of UDDI search is also defeated as whole web is searched for the user query word, so there is no more specification of searching by service name/category only. It takes less time as compared to a usual open source crawler which reads every word of each child link one by one in desktop application.

It provides a reliable and trust-worthy service discovery. And further it provides up to date information.

The organization of paper is such that Section 2 describes the previous research related to web service discovery. Section 3 presents the detailed overview of proposed framework including algorithm. Section 4 presents the implementation and key mechanism and Section 5 gives analysis of proposed framework. Finally, conclusion and future work is given in section 6.

2. RELATED WORK

A web service search engine [1] has its basis on the study that centrally maintained repositories are not enough to service search and keyword search does not provide full matching requirements for user query.

At first there is a focused crawling for WSDL. They have considered the information provided in WSDL documentation.

In a next step they have refined the results on user's explicit feedback from users. They used HeritrixWeb crawler by adding some rules to crawl only relevant pages. In next stage they removed duplicate results. However they could not achieve a relative accuracy in the retrieval.

A survey paper [2] has given very brief and interesting investigation of service discovery on basis of requirements given by the user. They say that WSDL document does not contain semantic descriptions of the service. So they do not provide non functional attributes of the service. For UDDI service discovery they raised a problem that it provides limited space for user to search on basis of keyword. As it only offer service name and category search. Also they indicated that most public UDDI's have been shut down. And there is no worldwide registry where all web services are published so there is no procedure to check performance and scalability. While describing middle agent challenges they narrated that WSDL documents contain lots of tags, which make it difficult for the agent to extract the information.

They concluded that WSDL handles functional requirements of a web service. An analysis of the various techniques used by search engines such as Google, Yahoo, and Web Crawlers has been provided to find their limitations.

Woogle [3] is a web service search engine. They have done extraction of information about wsdl functionality descriptions, inputs and outputs. They used clustering of parameters, matching of input output and operations, and stored the results in a database.

They compared their method with Func and Comb.

Comparison of words only with operation names is done by Func method. Whereas in Comb method web service names, parameters names and descriptions are also used for matching; in contrast to Woogle, both of the mentioned keywords are used.

In multi-registry environments THE WEB SERVICES RESPOSITORY BUILDER [4] provides foundation for web service discovery. It also provides reliability to some extent. A responsibility of crawler is that it actively seeks Web services; they made a registry monitor to track any changes of the provided registries. Further there is a Term Probing (TB) component which is responsible to extract words from WSDL descriptions, at end they provide web service storage to enable web service search. However there is no semantic support for service UDDI. They have used the specific registries such as MUBR, MUTR, SUBR and SUTR and they go around among them. So the framework is not flexible to be scaled.

The architecture in [5] extends SOA with Quality of service support for web services. In addition, it verifies, certifies, confirms, and monitors QoS properties. The architecture contains these major roles: - UDDI with QoS Information, Verifier and Certifier, Discovery Agent, QoS Matching, Ranking and Selection Algorithm. The discovery agent discovers functionally similar web service from provided UDDI registry when it receives request from the user.

They described main features required for a Qos based agent. Response Time, Availability, Throughput, Price are considered .Their approach is dynamic which keep cover on actual systems complexity. However their architecture is theoretical so there is no performance test. They argue that there framework will enable a more flexible, and trustable architecture.

Web services are XML based software components [6].

So they can be discovered in basis of signature and interface matching. So the search process depends on actual components of the service completely. WSDL is an XML based format which not only defines it functionality but also abstract operations and network bindings. [7].

Keyword matching is used for service discovery using UDDI. The work is matching XML schema with various comparisons using intelligent algorithms. Suffix, prefix and infix can be used for string matching. [8]

Liang-Jie Zhang, Qun Zhou [9] their framework solves the problem of linked documents. WSIL is used to search the chain services and results are return to the users after aggregation. So they solved the problem of manual link documents search. The chains of the documents are retrieved by re exploring the links in history using some calculations and caching.

Paul Palathingal [11] gave an agent based approach. The agent acts dynamically to discover, invoke and then execute the web services. Using agents it is possible that the sender never knows the receivers address. The agent who sends request for the service gets results from then the next agent; composition agent composes the web service.Service Profile method is used for

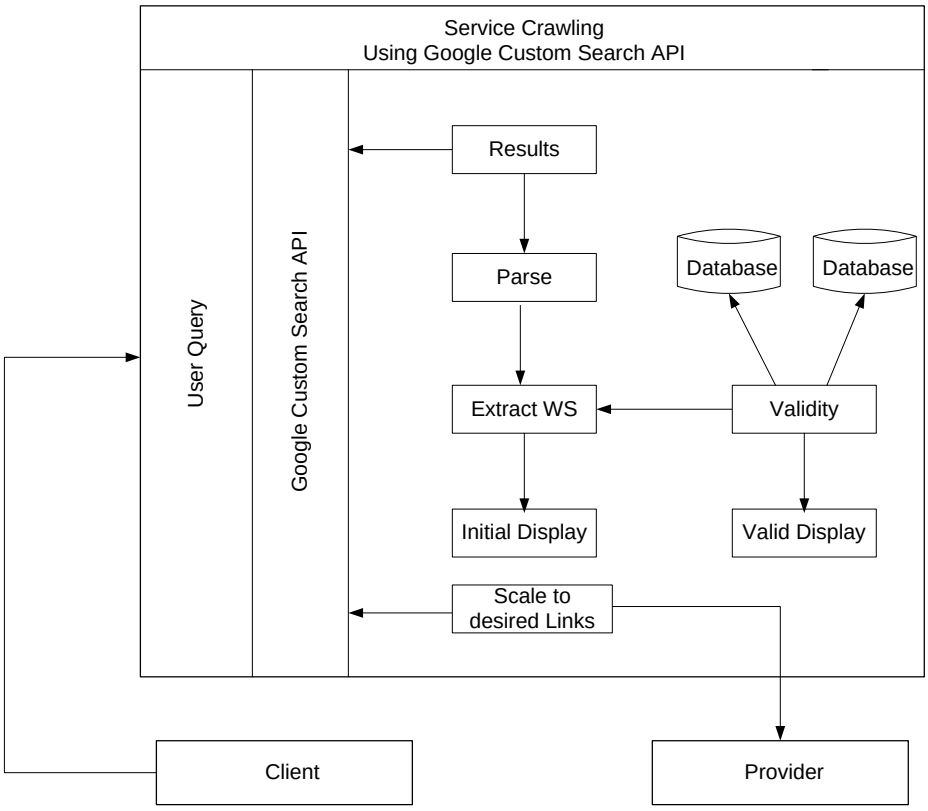

Fig 1. Framework for Service Crawling using Google Custom search API

Dynamic Web Service Discovery in [14]. They do not describe the internal web service behavior.

Lots of work has been done for web service discovery. All the frameworks provide best results in some way or other. But there is still a need for better discovery processes. Our paper gives an approach and proposes a framework that is flexible, scalable, reliable, and efficient.

3. PROPOSED FRAMEWORK

The framework shown in Fig 1 is quite simple and understandable. It includes following steps.

1. User queries the system. The input can be any word in users mind. The system matches the query word not only with service interface but also with its methods.
2. The request goes to Google Custom Search Engine through Google Custom Search API.
3. The engine has been scaled to the desired links to crawl. It can be scaled any time.
4. Engine crawls on all the links given and produces the results.
5. Results produced are not user understandable format. So the system parses the results produced.
6. System Extracts the Wsdl files from the set of results.
7. Results are displayed to the Client.
8. To check whether the service is available at given time. We have performed the validity check.
9. Results are displayed and sent to local database.
10. A backup database is maintained to provide reliability.

A. Pseudo Code: The pseudo code of proposed technique is given as:

Algorithm: Web Service Crawling
Input: Request for Web service
Output: Desired Service
Crawling links are added to Google Custom Search Engine;
User enters input request for web service;
For each input
    Input goes to Google Custom Search Engine through Google Custom Search API.
    Engine produces results.
    Results are parsed to human readable format.
    Only wsdl link and related information are extracted from the results.
    Results are displayed to user.
    Validity check is performed
    Valid results at present are displayed
    If result is not already in database
        Results are stored in database.
        Results are stored in backup database.
If no result found for user query word
    Message dialogue is displayed to enter synonym query word, Or to scale the engine to more links

Fig 2. Algorithm for Service Crawling using Google Custom search API

4. IMPLEMENTATION

The implementation has been done using Netbeans 6.9. Json is used along with google custom search api to get results of user query. JSON (JavaScript Object Notation) is a data-interchange format. JSON is completely independent of any language or environment but uses standards that are familiar to programmers. The specific format for the single JSON/Atom Custom Search API URI is:
"https://www.googleapis.com/customsearch/v1?parameters"
And the parameters we inserted are: "https://www.googleapis.com/customsearch/v1?key=INSERT-YOUR-KEY&cx=017576662512468239146:omuauf_lfve &callback=processResults &q=weather";

Where key is given to authenticate user, cx: The identifier of the custom search engine, callback is JSON Callback function to handle response. And q is actual query word. Being free user of the engine we can query 100 words per day. Can extend this limit by some payments required.

After getting the result from API into Net Beans we parsed the results to display only required information to user. Parsing required many matching and splitting statements. Next to parsing we have done wsdl extraction by matching end part resulting URL links to "asmx" or "wsdl". It is possible that when the user queries for a service the engine responds links that are not available at present time, i.e. timed out or any network error. We have performed validity check for that. To make the system reliable we maintained databases. MYSQL 5.5 is used to store information for future use. Only that information is stored which is not added to the database previously. Also we maintained a duplicate database to provide reliability. Following is responses message of JSON, we have only displayed two results of message.
Figure 3 shows response of JSON when queried through Google API. Figure 4 and 5 are GUI for Service Crawling through Google Custom Search API.


```
// API callback
processResults
(
{
 "kind": "customsearch#search",
 "url":
{
  "type": "application/json",
  "template":
"https://www.googleapis.com/customsearch/v1?q={searc
hTerms}&num={count?}&start={startIndex?}&hr={lang
uage?}&safe={safe?}&cx={cx?}&cref={cref?}&sort={s
ort?}&filter={filter?}&gl={gl?}&cr={cr?}&googlehost=
{googleHost?}&alt=json"
 },
"queries":
 {
 "request":
  [
 {
  {
"title": "Google Custom Search - .*weather.* ",
 "totalResults": "2",
   "searchTerms": ".*weather.*asmx?wsdl",
   "count": 10,
   "startIndex": 1,
   "inputEncoding": "utf8",
   "outputEncoding": "utf8",
   "safe": "off",
   "cx": "00138924657042:ocz3xgu",
   "filter": "1"
  }
 ]
},
 "context": {   "title": "Service Search"
 },
 "items": [
 {
  "kind": "customsearch#result",
  "title": "global weather wsdl - WebserviceX.NET",
  "htmlTitle": "\u003cb\u003eglobal weather
wsdl\u003c/b\u003e - WebserviceX.NET",
  "link":
"http://www.webservicex.com/globalweather.asmx?wsdl",
  "displayLink": "www.webservicex.com",
  "snippet": "Get weather report for all major cities around
the world. Get all major cities by   country name(full /
part). Get weather report for all major cities around the
world. ...",
  "cacheId": "R77gPNVFbxMJ"
  },
 {
  "kind": "customsearch#result",
  "title": "Global Weather - WebserviceX.NET",
 "htmlTitle": "\u003cb\u003eGlobal
Weather\u003c/b\u003e - WebserviceX.NET",
  "link":
"http://www.webservicex.com/ws/WSDetails.aspx?WSID
=56&CATID=12",
  "displayLink": "www.webservicex.com",
  "snippet": "Current weather and weather conditions for
major cities around the world ... http://
www.webservicex.net/globalweather.asmx?WSDL Demo
of this Web service ...",
  "cacheId": "D5o5Lqe8nGAJ",
  "pagemap": {
   "metatags": [
    {
     "code_language": "C#",
     "vs_defaultclientscript": "JavaScript",
     "vs_targetschema":
"http://schemas.microsoft.com/intellisense/ie5"
    }
 ]
  },
 );
```


Fig 3.JSON response Message, Crawling using Google Custom search API

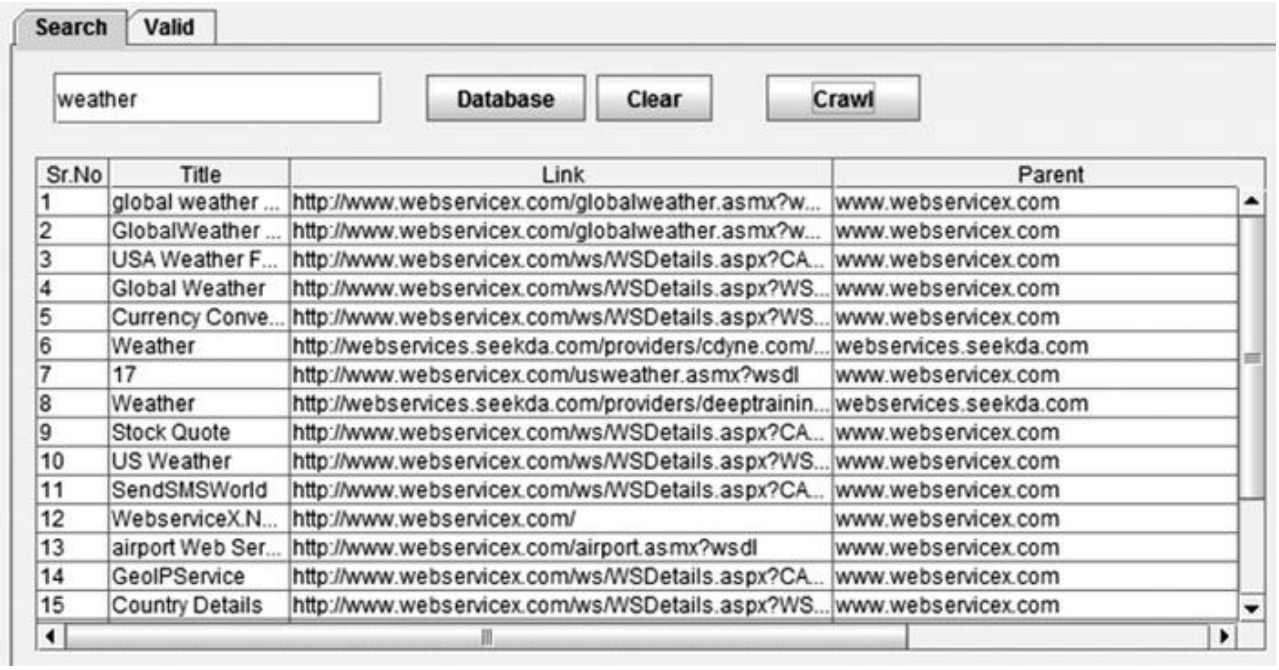

Fig 4.Crawl results

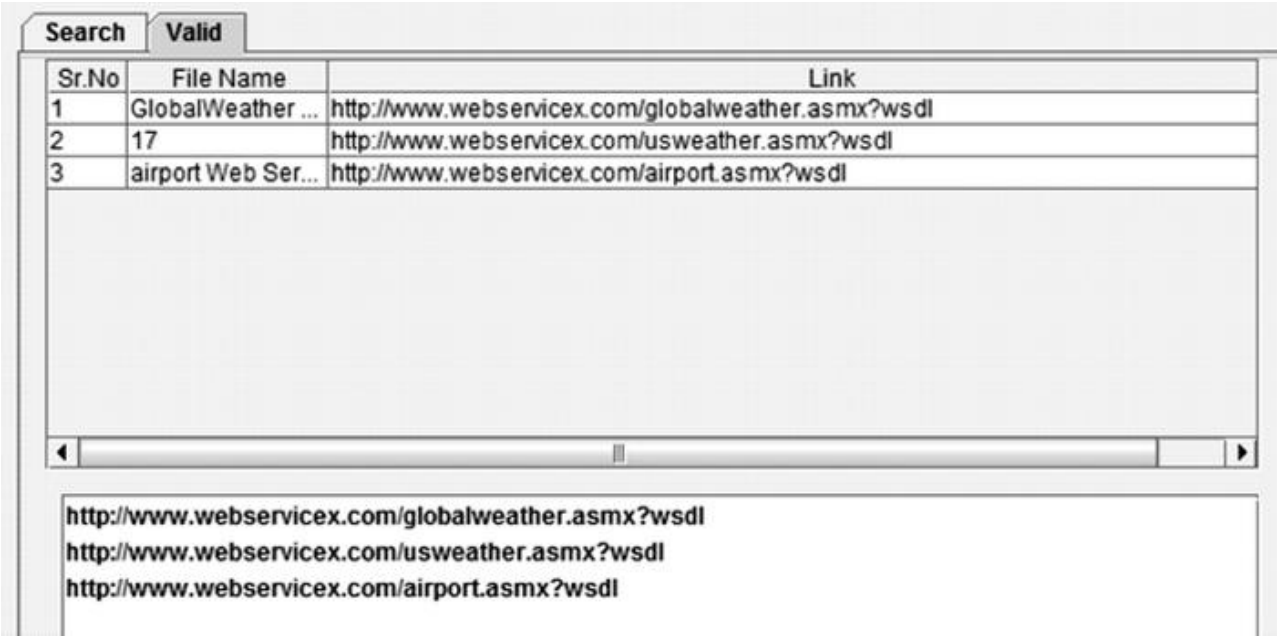

Fig 5.Valid WSDLS

5. EXPERIMENTAL EVALUATION

Since we can add more parent links to Google Custom engine, the user has more chances of getting the required service which is updated and exact. So our framework is scalable and flexible. Crawling a link is same as compared to other open source crawlers. The user query is matched on all the available child links of the provided link. But the engine response is efficient than those application crawlers.

Further those crawlers can crawl only one domain at a time. The custom search engine crawls all the provided links at once. We measured top-k precision (Pk) to check the overall performance. The formula we used is

$$pk = \frac{|retriverel_k|}{k}$$

Where k is total number of results retrieved and retriverel_k is total number of relevant results. [3]

6. RESULTS AND DISCUSSION

Applying above formula for results and analysis. We have taken average of 25 samples for each k precision. The results we got are shown in graph below Figure 6. Since we performed a check to extract only wsdl. And our system is matching the user query to both interface and functional level of wsdl. We got better top k precision as compared to [3] and two other naïve algorithms Func and Comb.

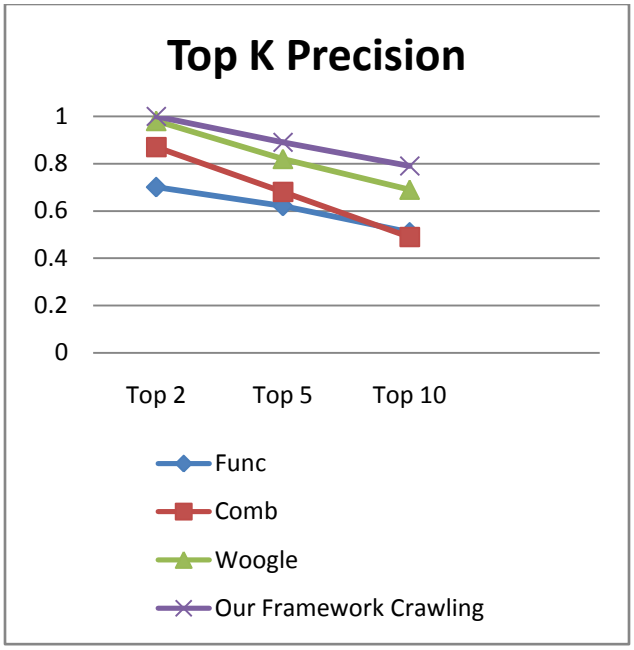

Fig 6.Top K Precision

7. CONCLUSION

This paper presents a framework for service crawling using Google Custom Search API. The framework is flexible, scalable, efficient and reliable.

In our approach the requester always gets up to date services the retrieval is fast and efficient. Also the client is able to add more repositories from where the services can be crawled. Our framework covered the limitations of formal UDDI search by searching whole page for user query. So user is not limited to give only the service name or category.Also it covers the limitation of usual crawlers in which the crawling for service can be done on only one domain at a time. We can crawl on heterogeneous registries.

Though there are many web service crawlers available online but our framework is for those clients who want to crawl and invoke services from a desktop applications. To provide reliability we have made a database to store the crawled services. To prevent duplication the system only adds those services which are not already present in the database. The updated information retrieval means the system checks weather the service is available at present or not. Also the results give better precision as compared to online engines for service search. Thus the proposed algorithm fix current issues of web services discovery. In future, the framework can be extended by making use of AI algorithms for discovery process. We will also experiment with Indexer discovery algorithm [15]. We plan to add ranking mechanism to index the links such that more trusted ones can be prioritized.

8. REFERENCES

[1] Holger Lausen and Thomas Haselwanter, "Finding Web Services" 2007.

[2] Mydhili K Nair, Dr. V.Gopalakrishna, "Look Before You Leap: A Survey of Web Service Discovery" International Journal of Computer Applications (0975 – 8887) Volume 7– No.5, September 2010

[3] Xin Dong Alon Halevy Jayant Madhavan Ema Nemes Jun Zhang, "Similarity Search for Web Services" Proceedings of the 30th VLDB Conference,Toronto, Canada, 2004

[4] Eyhab Al-Masri and Qusay H. Mahmoud, " Framework for Efficient Discovery of WebServices across Heterogeneous Registries", (NSERC), 2007

[5] T. Rajendran, Dr.P. Balasubramanie "An Optimal Agent-Based Architecture for Dynamic Web Service Discovery with QoS", 2010 Second International conference on Computing, Communication and Networking Technologies

[6] Eyhab Al-Masri and Qusay H. Mahmoud, "WSCE: A Crawler Engine for Large-Scale Discovery of Web Services" (ICWS 2007)

[7] Karastoyanova and A. Buchmann, "Components, Middleware and Web Services," Technische Universität Darmstadt, 2003

[8] E. Christensen, F. Curbera, G. Meredith, and S.Weerawarana, "Web Services Description Language (WSDL) 1.1," 2001.

[9] H. H. Do and E. Rahm, "COMA – "A system for flexible combination of schema matching approaches," presented at 28th VLDB Conference, 2002.

[10] Liang-Jie Zhang, Qun Zhou, Tian Chao "A Dynamic Services Discovery Framework for Traversing Web Services Representation Chain", Proceedings of the IEEE International Conference on Web Services

[11] Paul Palathingal "Agent Approach for Service Discovery and Utilization", Proceedings of the 37th Hawaii International Conference – 2004.

[12] Aabhas V. Paliwal "Web Service Discovery via Semantic Association Ranking and Hyperclique Pattern Discovery", Proceedings of the 2006 IEEE/WIC/ACM International Conference.

[13] Holger Lausen and Thomas Haselwanter "Finding Web Services".

[14] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In Proc. Of the Twelfth International World Wide Web Conference (WWW 2003), pages 331-339, ACM, 2003.

[15] Saba Bashir, M.Younus Javed,Farhan Hassan Khan, "INDEXER BASED DYNAMIC WEBSERVICES DISCOVERY" (IJCSIS) International Journal of Computer Science and Information Security,Vol. 7, No. 2, February 2010