Ontology Based Search Engine Enhancer

Viji Gopal, N.S. Gowri Ganesh

Abstract—**The success of web service is to interact with the applications of different domains. To achieve this interoperability, dynamic discovery of web services is essential. The repository of the web service needs an index of the currently available web services on the internet, which can be done by explicit publishing by the developers or by addition of the web services using crawlers. There is a chance that web services are indexed, but in real not available permanently or temporarily due to some reasons. We propose the use of OSIAN (Ontology based Service Index Annotator) to check the web services as availability which can be currently used by the service consumers. This will be of great use for the service consumers as the time taken to search for the web services among a large set of provisioned web services listed out in the repository will be reduced.**

Index Terms—availability, ontology, repository, web services

I. Introduction

A Web Service is a software component that is described via WSDL and is capable of being accessed via standard network protocols. It can support interoperable machine-to-machine interaction over a network. Web services allow clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol. SOAP, UDDI and WSDL are the three core elements of a web service. The client queries a UDDI registry [8, 9] for the service either by name, category, identifier, or specification supported. Once located, the client obtains information about the location of a WSDL document from the UDDI registry.

Repositories are a basic part of Web Services. They make it possible to find Web Services. Once a Web Service is found in a repository, the repository also describes how to use the Web Service. The owners of Web Services publish them to the UDDI registry. Once published, the UDDI registry maintains pointers to the Web Service description and to the service. The UDDI allows clients to search this registry, find the intended service and retrieve its details. These details include the service invocation point as well as other information to help identify the service and its functionality. Also UDDI enables companies to advertise the business products and services they provide, as well as how they conduct business transactions on the Web.

The rest of the paper is organized as follows. Section II gives the background of the paper. Section III presents the traditional architecture for Web service indexation and discovery. Ontology based Service Index Annotator is discussed in Section IV. Conclusions and future research directions are briefly discussed in Section V.

II. Background

In a normal keyword search the search results are based on match between keywords present in the description of the published services and the search string. Pure keyword based search fails to retrieve services which are described using synonyms of the search string. Moreover, singular/plural word forms used in the service description also affect the search result. The result will contain all the services that contain the word in their interfaces. This becomes a stumbling stone in bringing out meaningful results. So using concepts instead of words for matching seems to be a better idea.

A requestor will be looking for some web services that will match his criteria. An efficient search engine is supposed to come up with very useful and specific results such that the requestor can easily select one or more among the returned web services. The search engine gives the user the references to the selected web services, once the user selects one or more web services he can contact the vendors of those web services and acquire the service or services provided by them for his use. A search engine should have the maximum precision and recall to give the best results to the user. If most of the web service references reaching the user are dead or not properly working at that point of time, obviously the user will lose his trust on the web service repository. Here we try to improve the performance of the search engine by the use of ontology and by checking the availability of the web services before they reach the hands of the user.

A. Semantic Web

Current web which can be assumed to be the biggest global database lacks the existence of a semantic structure to keep the interdependency of its components and as a result the information available on web is mostly human understandable. Semantic web provides some languages that express information in a machine process-able format. This implies that we can take more benefit from their processing power. A huge amount of data is conceptually related, but much of these relationships still have to be kept

Manuscript received January 7, 08.

Viji Gopal is a Master of Engineering Student in R.M.K.Engineering College, Kavaraipettai, Chennai (e-mail: vijigopalakrishnan@gmail.com).

N.S. Gowri Ganesh is with the Center for development of Advanced Computing, Chennai (phone: 91-44-2461 0880; fax: 91-44-2461-0898; e-mail: nsgganesh@cdac.in).

in human memory and not stored in an understandable way for machines. Ultimate goal of Semantic Web is to create some smarter content which could be understood by machines. When the content is understood by machine, some assertions may come out of the content and new pieces of information will be produced.

A search engine handles queries to retrieve web services (Figure 1). It takes as input a set of input parameters of the web services and a set of output parameters of the web service. The search engine contains four parts: a Crawler, an Index, a query interface, and a result interface. The retrieved services are ranked and presented to the user via the result interface. The crawler discovers, analyzes and indexes semantic descriptions of Web services. The structure of the index allows the questions above to be answered, by indexing services according to the concepts they relate to, and according to their relations with other services.

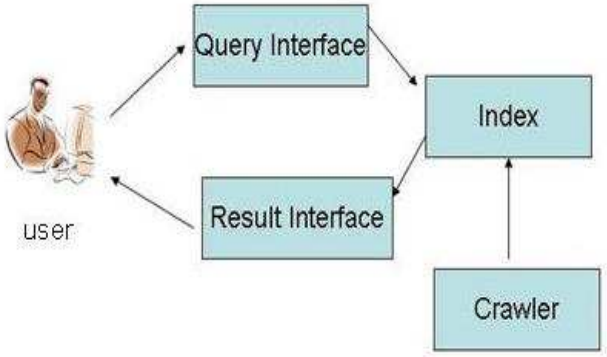

Figure 1 : The architecture of a search engine for web services

B. Ontology

In relation to computer science, ontology refers to computer-based resources that represent agreed domain semantics. Ontology consists of relatively generic knowledge that can be reused by different kinds of applications or tasks. Computer ontology is said to be an "agreement about a shared, formal, explicit and partial account of a conceptualisation". Domain rules restrict the semantics of concepts and conceptual relationships in a specific conceptualisation of a particular application domain. These rules must be satisfied by all applications that want to use ontology.

"An ontology is a formal explicit description of concepts in a domain of discourse (classes), properties of each concept describing various features and attributes of the concept (known as slots or roles or properties), and restrictions on slots (known as facets or role restrictions). Ontology together with a set of individual instances of classes constitutes a knowledge base." [2]. Semantic Web service technologies, such as the Ontology Web Language for Services (OWL-S), are developing the means by which services can be given richer semantic specifications. Richer semantics can enable fuller, more flexible automation of service provision and use, and support the construction of more powerful tools and methodologies.

C. Previous Works

There have been a number of research efforts along this track. This section deals with the research work carried out by the various researchers in the related area of the initial phase of software development. The problems encountered by various authors in their work and the proposed methods for the identification of those problems are discussed.

Colin Atkinson et.al [4] argues that one of the fundamental pillars of the web service vision is a brokerage system that enables services to be published to a searchable repository and later retrieved by potential users. This is the basic motivation for the UDDI standard. But this technology was not successful enough, and the few websites that today attempt to provide a web service brokerage facility do so using a simple cataloguing approach rather than UDDI.

Atkinson and Stoll [1] states that Service Oriented Architecture depends on the availability of accurate and universally understandable specifications of services. WSDL [3] is a mechanism which gives the description of the 'procedures' that a service offers. The information to invoke a service is available from its WSDL file. OWL-S [2,5] and some other languages define the semantics of services. But they do not change the underlying WSDL service specification. This principle has a drawback. Much of the information we need is context specific where WSDL specification is not context independent. Therefore it is good to specify a service using different abstractions which give a more precise description of context sensitive information. As long as there is a well defined mapping from the specification to one of the implementations, a specification can take any form.

Eran Toch et.al [6, 10] proposes a semantic web service search engine called Object-PrOcedure-SemanticS Unified Matching (OPOSSUM). It is shown that the main challenge in service-retrieval is the lack of semantics in their interface description for precise search. The semantic approach to service-retrieval is based on expanding the description of Web services with formal semantic models, such as OWL-S [9]. These models relate the services to concepts making it easy to retrieve them. In order to address the current limitations of service retrieval, they had developed OPOSSUM (Object-PrOcedure-SemanticS Unified Matching). It is a Web-based search engine that uses semantic methods for precise and efficient retrieval of Web services, based on their WSDL descriptions. OPOSSUM crawls the Web for WSDL descriptions, transforming them into ontological-based models of the Web services. More specifically, we propose an architecture that will facilitate the discovery of semantic Web services with availability checking.

III. TRADITIONAL SYSTEM

In a traditional system (figure 2), service providers describe the interface to their web service using WSDL, and publish their services in a UDDI repository by providing appropriate "meta data" such as provider identity

(white pages), a categorization of the provider's industry (yellow pages) and technical information necessary to invoke the service (green pages). Developers interested in using web services are then meant to be able to find components suitable for their needs by browsing the registry or using the keyword-based UDDI search facilities.

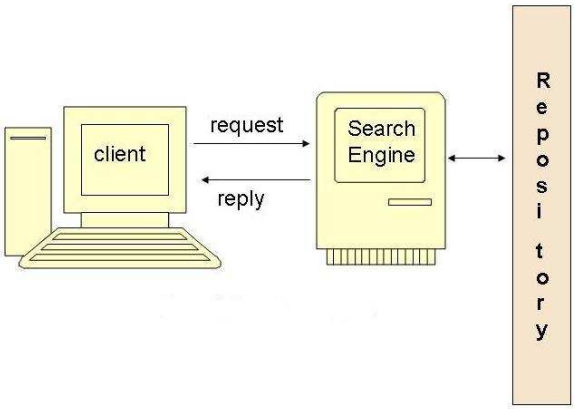

Figure 2 : Traditional system

To make the search much context specific and meaningful, semantic techniques can be made use of. A web crawler collects semantic web service descriptions from the ontology-oriented UDDI registries, Web sites hosting these services etc and creates and maintains a semantic web service repository. Once a service request is received from a client, the broker in the system can invoke a matching algorithm and a set of web service references are returned to the user.

IV. Ontology Based Service Index Annotator

A. The architecture

To improve the response time and reliability of web service repositories, we introduce a new strategy based on ontologies. Ontology based Service Index Annotator (OSIAN) is a module that can work in association with a search engine [11]. It acts as a mediator between the user and the search engine. Its main purpose is to ease the job of the search engine and to give quick results to the user. OSIAN comprises of two parts:

- An availability checker
- RTub (Recent Tub)

The availability checker checks whether a web service is still alive. If yes, it adds the service to the RTub. The web services which are in the RTub as well as in the search engine output will not be checked by the availability checker. RTub is a tub of web services which have been recently checked for availability. RTub is an ontology which has different classes for various domains, for example travel, conferences, pay rules etc. Based on the user input, the services from the corresponding domain are forwarded to the user.

Once a service is added to the RTub, it will automatically removed from the tub after a specified amount of time that is considered to be a reasonable time in which chances are very less for a web service to shut down. This time interval is called the lifetime of a service entry in the RTub. If at all a web service shuts down just after it has been entered in the RTub, it will remain there only for this time interval and it will be quickly removed when its life time expires.

OSIAN will act as a front end of the search engine. It acts as a mediator between the user and the search engine. But OSIAN will use the input user interface and output user interface of the search engine. Client posts the search criteria to the interface from where OSIAN absorbs it. Now OSIAN searches its RTub for matches. User can specify the input concept and/or output concept of the services that he needs. In RTub services are associated with their input and output concepts or ontologies. A search on the basis of the associated ontology is performed. If matches are available, it is returned to the output interface.

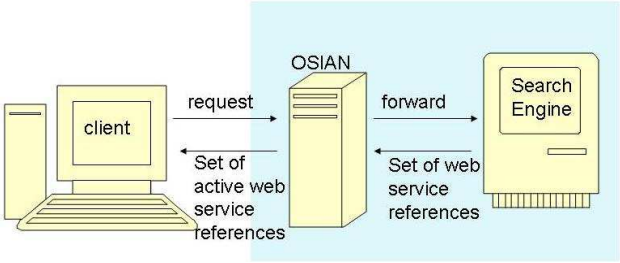

Figure 3 : The role of OSIAN in the Web Service Retrieval System

Else the search engine is invoked to search for matching web services in its repository. The search gives an output which is the input of OSIAN. Now OSIAN checks for the availability using its availability checker. The available web services are inserted in the RTub under the proper category. Now they are returned to the output interface so that the user can view them and make a selection from among the set of web services that he/she is presented with. Figure 3 shows how Ontology based Service Index Annotator connects the client to the search engine.

OSIAN has the following advantages:

- The system is able to automatically check the availability of the web services in the repository without user intervention
- The system is able to provide a cache effect when users access the services from the repository, making the response very fast
- The system is able to reduce the use of resources like processor time etc. by reducing the number of disk access needed
- The system is able to increase the overall performance of the search engine using it.

Figure 4 shows the structure of Ontology based Service Index Annotator system. We can test the availability of a potential web service by invoking one of its methods using randomly generated test data. This can be stored in a database and used for later periodic re-evaluation of the web service's availability. The receipt of any valid SOAP response can be interpreted as an indication that the service

is at least responding and can be regarded as being "live" [4].

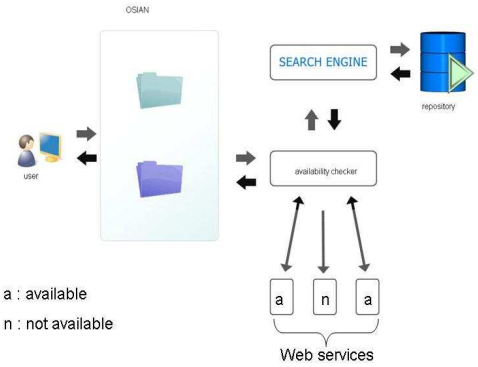

Figure 4: The architecture of OSIAN

When a user gives a query, we must search for specific operations that are similar rather than similar web service names. Even though the web service names are similar, their services may differ. So user comes to know whether a web service is of any use only when he checks it in detail. Two operations are considered to be similar if they take similar input, give similar output and the inputs and outputs have similar relationship between them. This brings out several web services which are of interest to the user. Parameter names in an operation can be grouped into meaningful concepts when a search is invoked. It improves the precision and recall.

Figure 5 is the block diagram of the structure of Ontology based Service Index Annotator system.

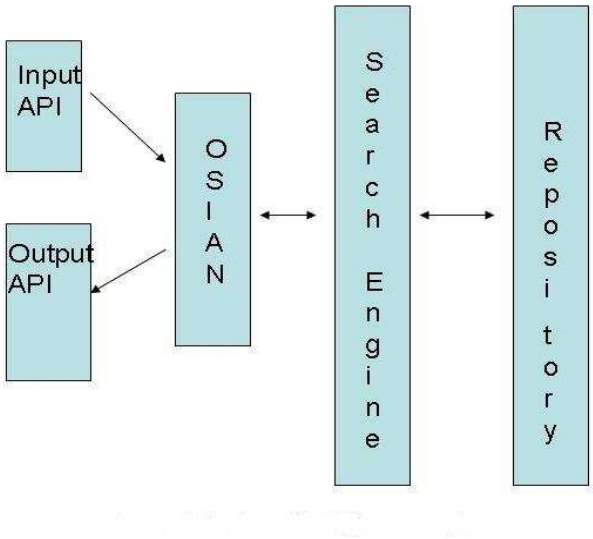

Figure 5: The Structure of OSIAN

B. A simple algorithm

A simple algorithm that describes the availability checking of OSIAN is as follows:

Let *x* be a web service reference returned by the search engine for the repository.
Input: a web service reference from the search engine
Output: the information that the web service is live or not.

Algorithm IV. 1

Get *x*
Check availability
If available
    Add in RTub
    Return "available"
Else
    Return "unavailable"

The attributes of the web service in RTub are the following:

1) Service
2) Associated concept
3) A timestamp
4) Type of the concept (input or output)

The *Service* is of type *Result*. *Result* is a class which indicates a single service returned by the search engine in response of a query. It contains functions to get the unique ID and name of a service, its WSDL link, description, rank etc.[6]

The timestamp is nothing but the time of availability check on the web service. The system time at the time of availability checking can be used as the timestamp. When the difference between this timestamp and the current system time is equal to the specified lifetime of the web service entry, the web service entry expires and is removed from RTub. This ensures the least number of dead or non-working web service references in the search result.

Concept is a string. It is one of the concepts to which the service is associated to. Type of the concept is an *Integer*. It indicates whether a concept is an input concept or an output concept to the web service.

We can illustrate it with an example. The user gives the input "*input : Car output : Cost"* to know the price of a car to purchase it. A normal search engine searches the repository using some matching algorithm and comes up with a number of web services which have *Car* and *cost* in its description. It may contain web services that calculate the market price of the car, the raw material cost of the car, the human resource cost involved etc. Some of them may be dead too. When the same input is given to Ontology based Service Index Annotator, it searches RTub and checks any web services are there under the *car-cost category* of *ontology* models in the RTub. If it finds such web services, that set is immediately returned to the user. As the web services have been checked for liveness recently, the user is sure to get live web services. If the matching web services are not available, the search is extended to the repository. A matching algorithm finds out matches and returns the result to the availability checker which in turn checks for the availability of each of the web services and returns only those that are *live*. This result is then updated in the RTub and forwarded to the user. Again

it is assured that all the services that reach the user are *live*. A web service *W* that was once removed from RTub can be placed in RTub at a later time when a user queries for that kind of web services and *W* is still *alive*.

C. Working of OSIAN: A swimlane diagram

The swimlane diagram of the message exchange in OSIAN is shown in figure 5. There are two cases to be considered: In case 1, we will describe the case where the RTub has the data required by the user. As it already has the data ready with it, it can directly supply the data to the user. The web services in the RTub need not be checked for availability. So the time for availability checking can be saved and the user gets quick reply.

In case 2, we describe the case where the RTub does not have the data required by the user. Now the control is handed over to the search engine and the usual search procedure is carried out. The result is passed to OSIAN and it checks the availability of the web services in the result returned by the search engine. Available web services get qualified to enter the RTub and to be displayed to the user.

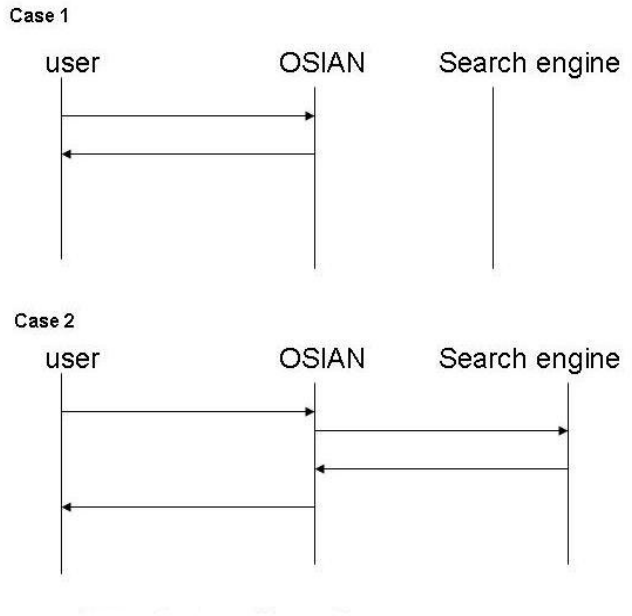

Figure 6 : Swimlane diagram of OSIAN's performance

OSIAN has the following advantages:

1. For a highly demanded area, OSIAN decreases the number of availability checking by a large amount
2. Almost all web services that reach the user are active. This increases the user's trust on the repository and avoids any wastage of time spent on non-working services.

V. STRUCTURE OF MAJOR SUBSYSTEMS

A. Service Retrieval Subsystem

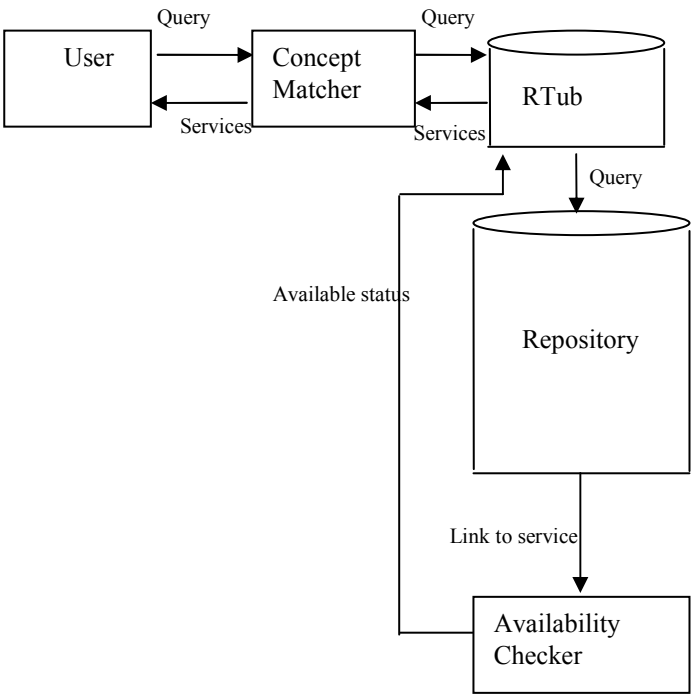

Figure 7. *Service Retrieval*

As shown in Figure 7, once a query has been given to the system, it checks its RTub. If matching services are found, it is immediately returned to the user. If matching services are not found, it goes and checks the tables on the disk (repository). The retrieved services are checked for availability and available services are added in the RTub. Next time when same query comes, it is directly answered back from the fast memory which stores the services for later use.

RTub contains instances of the class *RTubEntry*. Each instance has a unique ID, a service name, a concept associated with it, a type number showing if it's the input concept/output concept of the service and a timestamp which indicates the time of the last availability checking. The value of the timestamp is updated every time an availability check is performed.

B. Periodic Availability Checking

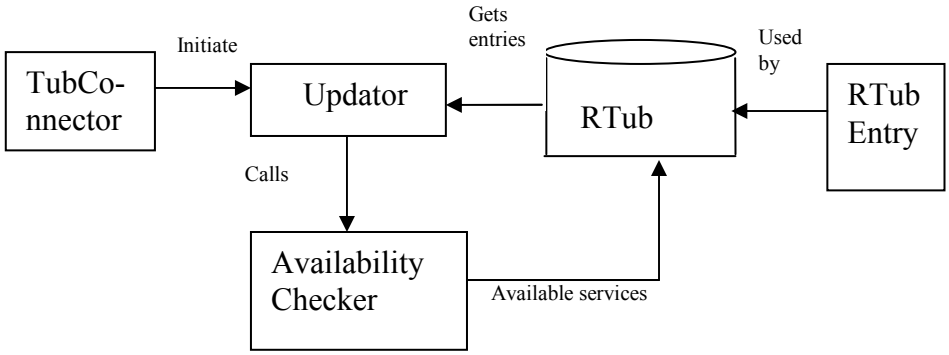

Figure 8. *Availability Checker*

As shown in Figure 8, the purpose of this feature is to detect the non-working services in the repository and remove them. Input is the link to the WSDL of the service and based on the output, service is retained in RTub or removed from RTub. TubConnector is a module that initiates Updator in a periodic basis. Updator module retrieves services in the RTub and invokes the AvailabilityChecker module. Available services are retained in the RTub, others are deleted.

VI. Experimental Evaluation

In this section, we evaluate our approach in two ways

1. Improvement in response time
2. Automatic availability checking

A. Improvement in Response Time

Evaluation was based on an implementation of OSIAN using Java and a MySQL server. A dedicated personal computer running Windows XP with 1.256GB RAM was used for all the experiments. The services and ontologies are all locally stored in the disk. When launched in World Wide Web, the time for network traffic will be added up. But this will be same for both the benchmark product and our product. So we can neglect it without affecting the performance comparison result.

We compared the response time values of OSIAN with those of OPOSSUM by running several queries. Our results show that we succeeded in improving the response time of the search engine compared to that of basic OPOSSUM. The basic search engine and the enhanced search engine vary considerably in query response time.

Table 1. *The query response time of OSIAN vs. OPOSSUM (measured in milliseconds)*

Query	*No .of time the query is evaluated*	*OPOSSUM*	*OSIAN*
Input: quotes and Output : greet	*1*	*14*	*40*
Input: quotes	*1*	*8*	*1*
Output : greet	*1*	*6*	*1*
Greet	*1*	*12*	*1*
Quotes	*1*	*9*	*1*
Greet	*2*	*15*	*1*
quotes	*2*	*13*	*1*
Input: quotes and Output : greet	*2*	*14*	*1*

Table 1 presents a comparison of response time of OSIAN and OPOSSUM to different queries. The results clearly show the benefits of a direct memory storage mechanism for answering repeated queries, which improves the performance of the query evaluation and response time.

From Table 1, it is clear that once result of a query has been stored in the fast primary memory then there needs no more disk reading or availability checking for the same query or for a similar query. The first time a query is given, OSIAN takes some time to check for the availability of those services which are in the result set, but not in the RTub. So it will be good for the service provider to run a query which will retrieve the newly deployed service. Once it has taken its place in the RTub, next time when a user gives a similar query, OSIAN will skip the time for availability checking. Of course all the above specified data is not constant whenever you run the machine. It changes slightly depending on the machine's current load. Nevertheless, OSIAN's response time has shown no changes around 95% of the time we ran the test (except for a query's first evaluation).

A service's availability is checked again in a specified interval when its *safe life span* is over. This is done by the availability checker. This process can be scheduled for

some low traffic time like midnight to avoid unnecessary delay.

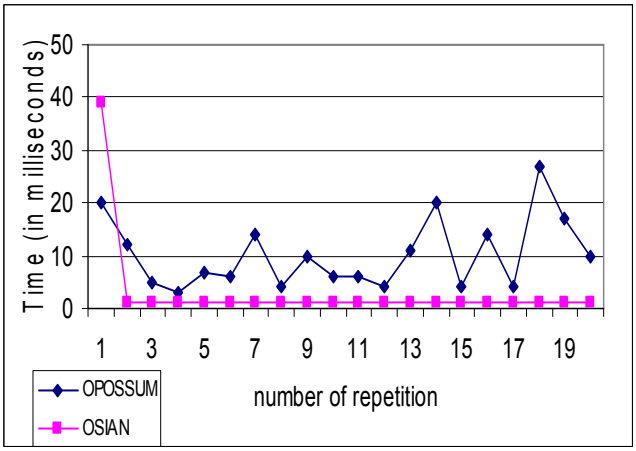

Figure 9: *Response Time Comparison for Multiple Executions of Single Query*

A query searching for all services which give *quotes* as output is given to both OPOSSUM and OSIAN. Figure 9 shows how each of them respond to the same query each time.

Figure 10 shows the response time of both on giving different queries searching for entirely different result sets. Five random queries were run 20 times each, the response time at each time was noted down and their average was calculated. The experiments show that OSIAN is particularly useful in databases which are queried for similar services. As number of repetitions increase, its efficiency increases.

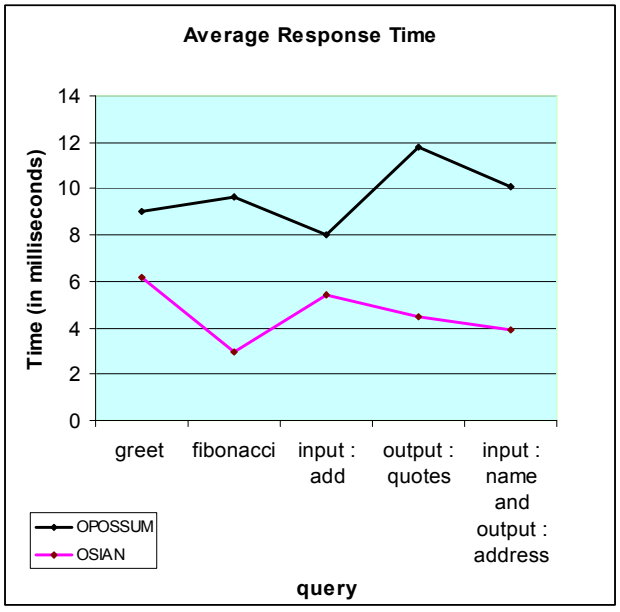

Figure 10: *Average Response Time*

B. Automatic Availability Checking

Automatic availability checking needed no comparison because we could not find any product in the market which does the same. So, only a console output of the background work has been shown here just to illustrate its working. It shows the console output when user gave the query "quotes" two times in a row. In real, this part is not visible to the user.

QUERY FROM PARSER : quotes

The query took 39 milliseconds

final content given is : QuoteOf Day
final content given is : publish quotes
final content given is : NiceQuoteService
****
Initiating the scheduler
****
- Job execution threads will use class loader of thread: http-8080-Processor25
- Quartz Scheduler v.1.6.0 created.
- RAMJobStore initialized.
- Quartz scheduler 'DefaultQuartzScheduler' initialized from default resource file in Quartz package: 'quartz.properties'
- Quartz scheduler version: 1.6.0
- Scheduler DefaultQuartzScheduler_$_NON_CLUSTERED started.

Entered the scheduled class
Services in RTub : 3
service : QuoteOf Day lifetime : 763
service : publish quotes lifetime : 760
service : NiceQuoteService lifetime : 760

QUERY FROM PARSER : quotes

The query took 1 millisecond

final content given is : QuoteOf Day
final content given is : publish quotes
final content given is : NiceQuoteService

Entered the scheduled class
Services in RTub : 3
service : QuoteOf Day lifetime : 45743
lifetime > safe life span
now deleting : RTubEntry@103c29b
Service available..!!!
service : publish quotes lifetime : 45740
lifetime > safe life span
now deleting : RTubEntry@1dd7bc5
Service available..!!!
service : NiceQuoteService lifetime : 45740
lifetime > safe life span
now deleting : RTubEntry@1e8c585
Service available..!!!

II. Conclusions and Future Research

There are many service search engines available to search and retrieve web services. The traditional system depends on keyword matching of service descriptions which has been proved to be very inefficient for service retrieval. Those who use semantic strategies have clearly shown an improvement in their precision and recall. OPOSSUM is one such search engine which has been selected as a platform for this project OSIAN. None of the present service search engines do any availability checking which is an important factor that decides the reliability of the repository and thereby user satisfaction. OSIAN is a module that works with a search engine improving its reliability and response time by using an availability checker and a fast memory which keeps track of recently used services. It ensures that the number of disk accesses is minimum, which saves a lot of time while searching for services. Similar queries have been run in both OPOSSUM and OSIAN and the results were compared. Our results show that we succeeded in improving the response time performance compared to those of OPOSSUM. Also OSIAN's availability checker checks for the availability of services before they are delivered to the user. This ensures that all the services reaching the user's hands are in proper working condition. It improves user satisfaction which is an important criterion of evaluating today's competitive business world. Increased user trust on the search engine justifies the overhead of adding a verifier. It is proved that OSIAN can improve the performance by a great deal by giving a cache effect to the retrieval part of the search engine and by automatically checking the availability of the services.

This work can further be enhanced in the following ways:

Even when a rich set of ontologies is built the ontology engineering process is not terminated. The next problem we should deal with is the issue of ontology aging. Ontologies are concepts. Concepts change as time changes. Changes can happen due to changing inference rules, changing semantic rules and arrival of new ontologies. Results, extracted from an out-of-dated ontology can not be used in a totally meaningful way. There should be some mechanism to detect ontology aging and to update it time to time.

Multilingual support is recognized as one of the most important challenges of Semantic Web. Nowadays English is the predominating language and about 70 percent of Internet content is in English, but only about 44 percent of Internet users are native English speakers. Especially in India, this issue is quite essential and the diversity of languages needs to be taken into consideration. One method to handle this problem is by establishing relevant inter-ontology translators that map ontologies and content to other languages.

Also we are planning to extend the current research such that the search engine can identify trusted service providers because trustworthiness is something that is given very high importance in today's business world.Another issue that comes to light is that UDDI limits the service discovery only to functional requirements. Different web services that satisfy the same requirements can have different Quality of Service (QoS). We can improve the service discovery if we can retrieve services with high QoS. We plan to incorporate this concept as our future work. We can represent QoS specifications using ontology based on XML which enables services to be matched semantically and dynamically. This will allow the user to match the provider's advertised value of a QoS to their own preferences.

References

[1] Colin Atkinson, Philipp Bostan, Oliver Hummel and Dietmar Stoll. *A Practical Approach to Web Service Discovery and Retrieval*. 2007 IEEE International Conference on Web Services (ICWS 2007). July 2007.

[2] Birgit Hofreiter, Christian Huemer, Wolfgang Klas. *ebXML: Status, Research Issues, and Obstacles*. Proceedings of the 12th Int'l Workshop on Research Issues in Data Engineering: Engineering e-Commerce/ e-Business Systems (RIDE'02).

[3] Atkinson, C. and Stoll, D. and Acker, H. and Dadam, P. and Lauer, M. and Reichert, M.U. (2006) *Separating Per-client and Pan-client Views in Service Specification.* In: Proceedings of the 2006 International Workshop on Service-oriented Software Engineering, 27 - 28 May 2006, Shanghai, China. pp. 47-53.

[4] Eyhab Al-Masri and Qusay H. Mahmoud. *Interoperability among Service Registry Standards*. Published by the IEEE Computer Society. IEEE INTERNET COMPUTING. May-June 2007.

[5] Joseph Chiusano (Booz|Allen|Hamilton). *UDDI and ebXML Registry: A Co-Existence Paradigm*. March 2003.

[6] Eran Toch, Iris Reinhartz-Berger, Avigdor Gal, and Dov Dori. *OPOSSUM: Bridging the Gap between Web Services and the Semantic Web.* Proceedings of Next Generation Information Technologies and Systems. July 4-6, 2006. pp. 357-358.

[7] Natalya F. Noy and Deborah L. McGuinness (2001) *"Ontology Development 101: Guide to Creating Your First Ontology"* http://protege.stanford.edu/publications/ontology_development/ ontology101.html

[8] Farquhar, A. (1997). Ontolingua tutorial. http://ksl-web.stanford.edu/people/axf /tutorial.pdf

[9] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein,Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, Katia Sycara. *Bringing Semantics to Web Services: The OWL-S Approach.* www.daml.org/services/owl-s/OWL-S-SWSWPC2004-CameraReady.doc

[10] Wenli Dong. *QoS Driven Service Discovery Method Based on Extended UDDI*. Third International Conference on Natural Computation (ICNC 2007). pp. 317-324

[11] Xin Dong Alon Halevy Jayant Madhavan Ema Nemes Jun Zhang. *Similarity Search for Web Services.* Proceedings of the 30th VLDB Conference, Toronto, Canada, 29 August - 3 September 2004. Pages: 372 – 383

[12] Eran Toch, Avigdor Gal, Iris Reinhartz-Berger, and Dov Dori, *A Semantic Approach to Approximate Service Retrieval*. ACM Transactions on Internet Technology, Vol. 8, No. 1, pp. 2:1-2:30, November 2007.

[13] Berners-Lee, T., Hendler, J., Lassila, O., The Semantic Web, Scientific American, 284(5), 2001, pp. 34-43