COMPARATIVE STUDY OF MECHANISMS FOR DISCOVERING THE MOST APPROPRIATE WEB SERVICE AND PROPOSING AN EFFICIENT WEB SERVICE DISCOVERY MECHANISM

A thesis submitted to

Tilak Maharashtra Vidyapeeth, Pune

For the Degree of Doctor of Philosophy (Ph.D.)

In

Computer Management

Under the Faculty of Management

Submitted By
Netra Patil

Under the supervision of
Dr. Arpita Gopal

Director – MCA

Sinhgad Institute of Business Administration and Research,
Kondhwa-Bk, Pune - 411048

October 2014

Declaration

I hereby declare that the thesis entitled **"Comparative Study Of Mechanisms For Discovering The Most Appropriate Web Service And Proposing An Efficient Web Service Discovery Mechanism"** completed and written by me has not been previously formed the basis for the award of any Degree or other similar title upon me of this or any other University or examining body.

Place : Pune	(Mrs. Netra Patil)
Date : 16-Oct-2014	Research Student

Certificate

This is to certify that the thesis entitled **“Comparative Study Of Mechanisms For Discovering The Most Appropriate Web Service And Proposing An Efficient Web Service Discovery Mechanism”** which is being submitted for the degree of Doctor of Vidyavachaspati (Ph. D.) in Computer Management to Tilak Maharashtra Vidyapeeth is an original piece of research work completed by Mrs. Netra Patil under my supervision and guidance.

To the best of my knowledge and belief the work incorporated in this thesis has not been formed the basis for the award of any Degree or similar title of this or any other university or examining body upon her.

Place : Pune

Date : 16-Oct-2014

Dr. Arpita Gopal

Director-MCA

Sinhgad Institute of Business Administration and Research, Kondhwa(Bk), Pune-411048

Acknowledgement

Pursuing a PhD is a truly life-changing journey that is not possible to accomplish without the help, support and guidance of many people. The most pleasurable aspect of writing a research thesis is the opportunity it provides to thank God and express heartfelt gratitude to all those who have been a great source of encouragement and enlightenment to complete each and every task with utmost passion and dedication.

First of all, I would like to express my deepest gratitude to my guide, **Dr. Arpita Gopal**, Director-MCA, Sinhgad Institute of Business Administration and Research, Kondhwa, Pune, India for sparing her most valuable time for continuously motivating and guiding me during the course of research work. She gave me the opportunity to perform and accomplish the biggest step towards an academic or research career. Her encouragement to explore new directions while maintaining focus has kept me productive and excited about research. Every moment spend with her was a profoundly enriching experience.

No amount of words can express the gratefulness for **Prof. M. N. Navale**, President, Sinhgad Technical Education Society, Pune, for his exemplary patronage to scholarly pursuits in granting me days off to enable me complete my research work and accommodating the research devoted hours so generaously.

I am also blessed with wonderful friends in many ways, my successes are theirs, too. Special thanks to my dear friend Dr. Chandrani Singh, Joint Director-MCA, Sinhgad Institute of Business Administration and Research, Kondhwa, Pune whose stimulating encouragement, and valuable suggestions have greatly contributed to this thesis and helped me round off the research successfully. All my friends gave me a lot of support and encouragement through the hard times. They deserve special acknowledgements - Thank you Rithambara Korpal, Rubina Sheikh, and many more friends from SIBAR.

I owe my sincere gratitude and a very special thanks to my parents and in-laws for bearing me through the thick and thin of my research tenure. They have always given me their unconditional love and moral support. I am indebted to them for inculcating the love for knowledge in me. If it wasn’t for their support, co-operation and encouragement, this endeavor would not have been possible.

I would take this opportunity to thank and acknowledge my life partner, Dr. Prashant Patil**,** for his continuous encouragement and cooperation throughout completion of my research work. He has always been there in both the good and the bad moments and gave me the power and inspiration to move on and fight for my goals.

And last but not the least, to my dearest daughter Tejas, I offer my immense gratitude for her unconditional love, patience and tolerance. Thank you my darling for making it all worthwhile!

Contents

a. "Assessment of UDDI and ebXML Registry for e-Business Application" by Netra Patil, Dr. Arpita Gopal, in International Journal of Computer Science and Application, ISSN 0974-0767, Issue-III, December 2012.

b. “Comparative Study of mechanisms for Web Service Discovery based on Centralized approach focusing on UDDI” by Netra Patil, Dr. Arpita Gopal, in International Journal of Computer Applications, ISSN 0975-8887, January 2011.

c. "Enhancing UDDI registry for storing Qos in tModel for discovering web services" by Netra Patil, Dr. Arpita Gopal, in International Journal of Computer Science and Application, ISSN 0974-0767, Issue-I, January 2011.

d. “Ranking Web-services based on QoS for best-fit search” by Netra Patil, Dr. Arpita Gopal, in International Journal of Computer Science and Communication, ISSN 0973-7391, Volume-I, Number-II, September 2010.

e. “Quantifying Web Services on Quality Parameters for Best-fit Web-service Selection” by Netra Patil, Dr. Arpita Gopal, in International Journal of Computer Science and Application”, ISSN 0974-0767, Issue-II, January 2010.

f. “Comparative Study of Centralized and Decentralized Approaches for Web Service Discovery Mechanism”, by Netra Patil, Dr. Arpita Gopal, in International Conference IACC 2010 at Thapar University, Patiala.

g. “Model proposed for the senior management of an organization for utilizing resources effectively to adopt web services”, by Netra Patil, Dr. Arpita Gopal, in International Conference ICDM 2008 at IIM Ghaziabad, Delhi.

List of Figures

List of Tables

Chapter 1

Introduction

Web Service Technology (WST)- A Brief Concept

Companies have reorganized businesses using technology advents such as web-enabled business. These businesses have gotten highly promoted due to the ease with which application-to-application communication happens over the internet, the underlying framework being strong support of web service technology.

The fundamental concept is simple – web services allow us to make Remote Procedure Calls (RPCs) against an object over the Internet or a network. Web Services Technology is not the first of its kind to allow us to do this, but it differs from other technologies in its use of platform-neutral standards. For example HTTP and XML allow us to hide the implementation details entirely from the client. The client needs to know the URL of the service, and the data types used for the method calls, but don't need to know whether the service was built in Java and is running on Linux, or is an ASP.NET web service running on Windows. [97]

A Web service comprises of loosely coupled software components published, located and invoked across the web. A Web service is a means of performing distributed computing. A web service provides either some business functionality or information to other applications through an internet connection. For example,

- A recruiting company is interested in publishing its latest job-openings as a Web Service. Job placement (contracting) companies could be potential subscribers to this Web Service.

- An airline reservation system is interested in publishing its latest airfares as a Web Service. Travel agencies could be potential subscribers of this Web Service.

A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be found out by other software systems. These systems may then interact with the Web service in a way specified by its definition, using XML based messages conveyed by Internet protocols.[98]

1.1 Service Oriented Architecture (SOA) and Web Service Discovery

Since businesses have reorganized using the technology, there came a need to have architecture for building business application known as Service Oriented Architecture.

A Service Oriented Architecture (SOA) is architecture for building business applications as a set of loosely coupled black-box components organized to deliver a well-defined level of service by linking together business processes. One of the most important aspects of SOA is that it is a business approach and methodology as much as it is a technological approach and methodology. With SOA, the important business processes such as generating an invoice, calculating an interest rate, converting currency become business services. A business service is a sealed container of software code that describes a specific business process that can be connected to other business processes. One single business service for a given functionality can be used everywhere in the organization and whenever a business policy is changed, it is required to make change at only one place as the same service is used everywhere.

SOA can make it easier and faster to build and deploy IT systems that directly serve the goals of a business. SOA adds predictability and regularity between business rules, policy and software services. Therefore, one of the greatest selling points for SOA is that it can help management know what tasks a particular service is executing and what rules and policies are codified within these services. Being able to track this not only makes software within the company better but also makes corporate governance more predictable and less cumber some.

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Service-Oriented Architecture is a business-driven IT architecture approach that supports integrating your business as linked, repeatable business tasks, or services. SOA helps today's business innovate by ensuring that IT systems can adapt quickly, easily and economically to support rapidly changing business needs. SOA helps customers increase the flexibility of their business processes, strengthen their underlying IT infrastructure and reuse their existing IT investments by creating connections among disparate applications and information sources.

Service-oriented architecture is not a new concept. The first service-oriented architecture for many people in the past was with the use of DCOM or Object Request Brokers (ORBs) based on the CORBA specification. In these traditional distributed architectures, web services were used to facilitate point-to-point solutions. Hence, web service discovery was not a common concern.

The increasing number of web services available on the web raises a new and challenging problem, the location and discovery of these services. The lack of a proper discovery mechanism is hindering the potential of these technologies.

The growing numbers of web services descriptions are difficult to manage in open environments such as in the Web. The main problem arises due to the fact that hundreds of different web services exists providing thousands of different functionalities. They are built independent of each other at different locations by different people. Discovering a web service that matches the user's requirement is time consuming and tedious.

As the demand for web service consumption is rising, a series of questions arise concerning the methods and procedures to discover the most suitable web service to use. Web service discovery is the process of finding the most appropriate web services needed by a web service requestor.

There is a need for dynamic discovery mechanism that will be always up-to-date providing efficient and available web service choices.

Web service discovery mechanisms have a role even more important than web searching, because they facilitate the need for collaboration among various business processes and consumers over widely accepted web standards.

In the beginning of service-oriented computing, finding relevant web services was mainly done by searching through services registries (i.e. UDDI Business Registries or UBRs). Automated web service search engines were not necessary when web services were counted by the hundreds. However, the number of service registries is gradually increasing and web service access points (i.e. WSDLs) are no longer a scarce resource as there are thousands of web services scattered throughout the Web.

Business organizations need to advertise their services in a global environment to potential trading partners and they should also have a way to discover and interact with each other. Service consuming client must be able to find proper web services with less effort than currently required.
As web services have begun to expand across the internet, users need to be able to efficiently access and share web services. Production and interoperability of larger number of web services have lead to the emergence of new standards on how services can be published, discovered or used. Hence, mechanisms are required for efficient selection of appropriate web service instance in terms of quality and performance factors during web service consumption.

1.2 Approaches towards Web Service Discovery

Web service discovery is "the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria." [97] The goal is to find an appropriate Web service.
Under *manual discovery,* a requester *human* uses a discovery service (typically at design time) to locate and select a service description that meets the desired functional and other criteria.

Under *autonomous discovery,* the requester *agent* performs this task, either at design time or run time. The steps in discovering a web service are same in both cases. Only few issues such as interface requirement's need for standardization and trust have to be considered in this case, as the discovery is automated.

One situation in which autonomous discovery is often needed is when the requester agent has been interacting with a particular provider agent, but for some reason needs to refresh its choice of provider agent, either because the previous provider agent is no longer available, or other reasons.

There are three main approaches [97] for discovering a web service: as a registry approach, as an index approach, or as a peer-to-peer approach. Their differences and purposes are discussed below.

1.2.1 The Registry Approach

A registry is an authoritative, centrally controlled repository of services information. Service provider must publish the service information into the registry before that information is available to the service consumers. The registry owner decides who has authority to publish and update the service information into the registry. A company is not able to publish and update the information of services provided by another company. The registry owner decides what information can be published in the registry. Others cannot independently add to that information. UDDI is an example of the registry approach, but it can also be used as an index.

1.2.2 The Index Approach

An index is a collection or guide to information published by the service provider and that exists elsewhere. It is not authoritative and information that it references is not centrally controlled. In the case of an index, the service provider describes the service and functional descriptions on the Web, and the index owners collect them without service providers knowledge. Anyone can create their own index. When descriptions are exposed, they can be collected using web spiders and arranged into an index. Multiple organizations may have such indexes. The information contained in an index could be out of date. The information can be verified before use. Different indexes provide different kinds of information — some richer, some sparser. Google is an example of the index approach.

The key difference between registry and index approach is one of control: Who controls what and how service descriptions get discovered? In the registry model, it is the owner of the registry who controls this. In the index model, since anyone can create an index, market forces determine which indexes become popular.

1.2.3 Peer-to-Peer (P2P) Discovery

Peer-to-Peer (P2P) computing provides an alternative that does not rely on centralized registries and allows Web services to discover each other dynamically. At discovery time, a service requester queries its neighbors in search of a suitable Web service. If any one of them matches the request, then it replies. Otherwise each queries its own neighboring peers and the query propagates through the network until a particular hop count or other termination criterion is reached.

Peer-to-peer architectures do not need a centralized registry, since any node will respond to the queries it receives. P2P architectures do not have a single point of failure, such as a centralized registry. Furthermore, each node may contain its own indexing of the existing Web services. Finally, nodes contact each other directly, so the information they receive is known to be up-to-date. On the contrary, in the registry or index approach there may be significant latency between the time a Web service is updated and the updated description is reflected in the registry or index. The reliability provided by the high connectivity of P2P systems comes with performance costs and lack of guarantees of predicting the path of propagation. Any node in the P2P network has to provide the resources needed to guarantee query propagations and response routing, which in turn means that most of the time the node acts as a relayer of information that may be of no interest to the node itself. This results in inefficiencies and large overhead especially as the nodes become more numerous and connectivity increases. Furthermore, there may be no guarantee that a request will spread across the entire network, therefore there is no guarantee to find the providers of a service.

Further to the above approaches, justification lies in portraying the issues related to these approaches.

1.3 Issues in Web Service Discovery

In today's global world, every person is looking for cost and time effective services, which can give him/her satisfaction. Thanks to technological development because of which the world has come closer. There are number of software/IT companies which are providing web based services to global customers. Right from travel booking to buying and selling anything, customers do visit web portals very often. Based on the cost-benefit analysis customer makes selection and try to avail the services. On the other hand service provider companies (web portals) in association with IT companies, who develops the services, make efforts to meet the customers' needs and to satisfy them. However due to technical and non-technical problems, service providers as well as IT companies do find that customers have genuine complaints or grievances which they can or can not solve immediately. Because of this, loosing customers has become a great loss to the service providers. In order to solve this problem, this research has aimed to develop a model of efficient web service discovery mechanism. This will lead to help service engineers of service provider companies and ultimately general customers in making an effective search while logging onto the site for expected service based on certain parameters which will automatically make discovery by giving ranking/priority for cost-effective solution.

Web service discovery based on the non-functional aspects (e.g Quality of Service) has become a very important step to help service requestor to locate a desired service. Generally there are two types of service requestors – the human user who will use the services in complex application development or program which automatically sends request and select services for further processing. Many researchers are proposing various models, QoS description languages and frameworks for discovering and selecting an appropriate web service. However, from the literature study some issues which arise and need to be addressed are as -

- The end user's view has not been focused in their designs and the user support is either missing or lacking in these systems. Without the proper user support, the accuracy of the QoS requests cannot be guaranteed, and without accurate QoS requests, even the best selection model cannot satisfy users' requirements. Hence there is a need of a user oriented service selection system, which is important mainly for the human-involved service selection.

- An assumption that users can formulate requests which precisely reflect their QoS requirements may not be true as a user may not have the knowledge about what the realistic QoS values are. Also if the user requests for a service with randomly picked number for reliability as ‘greater than 95%’, the result could be zero matching services. Decreasing this number by a few percent, we may find some matching services. Because of this kind of difficulty of choosing a right number, it is not reliable for a selection system to assume the accuracy of the QoS requests from users. It is very advantageous if the selection system can assist users to choose the right QoS values.
- In many current systems, the user interface design is not given much importance. Different selection models are proposed and then it is assumed that users would have the ability to submit a proper query which will yield appropriate results using the model. The user may need to have the knowledge on ontology, utility functions etc. In reality, many of the users don’t have this kind of knowledge. So we should have a simple and a carefully designed interface to help users formulate the service request.
- With current QoS query languages, requestors may not be able to define their requirements in a precise and comprehensive way. For instance, many times the QoS requirement is represented as either a number (e.g. reliability: 95%), or a fuzzy description (e.g. reliability: very good). However, it is also possible that users may have a mixed request – numeric values on some QoS attributes and fuzzy expressions on others. Therefore, the selection model should have the ability to support this kind of request.
- Another issue we want to address is the lack of support for defining preference order on QoS attributes, e. g. which quality attributes should be given higher priority if there are more than one services satisfying all the criteria. Hence, it is necessary to define a separate preference order for QoS attributes, which is lacking in many current works.

1.4 Research Hypothesis

A number of web portals are offering web services for customers all over the world. Customers make selection of these web services based on certain parameters of their choice. However, there are certain loopholes in the mechanism of efficient web services discovery.

The proposed research work aims to develop an efficient model for web services discovery mechanism, which will wipe off the weaknesses in the existing web service architecture to satisfy the customers. The proposed model will assist in retrieving web services with desired functionality and provide a flexible tool which will guide the user to choose the right QoS parameter values, formulate precise requirements for these QoS parameters and define QoS parameters preference order or priorities for minimizing the search. The tool will rank the services based on the search criteria specified by the user and thus the most appropriate web service for the user will be found out using the proposed mechanism.

This research work intends to accomplish the following:

Given a list of web services with the similar functionality and different QoS values, this study aims at

1) Proposing a new discovery technique to store and manage QoS information of web services in the registry for ranking and finding the most appropriate web service from the list of published web services in the registry.
2) Designing a Web Service Discovery tool which will assist in –
    a. Publishing web services along with QoS information in UDDI registry.
    b. Requesting web services by specifying functional, QoS and Monitoring requirement along with the priority of QoS.
    c. Extracting monitor score from the service monitor which monitors the services at regular intervals for verifying advertised QoS by the service providers.
    d. Assigning weights to QoS and monitor scores as per the users preference and find the overall score of each service which are functionally matched.

   e. Ranking the services based on the calculated overall scores and return specified number of top ranked services to the user.

3) Comparing the proposed technique with existing techniques mainly on the relevance of quality of the results which is evaluated based on the degree of similarity between results obtained from the new technique and that of the existing one.

4) Based on the results obtained from above, implementing a web discovery tool with user-centric interface for discovering the most appropriate web service.

1.5 Research Objective

As a large number of web services are proliferating across the internet, end users or client applications need to be able to efficiently access and share web services. Production and interoperability of larger number of web services have lead to the emergence of new standards on how services can be published, discovered or used. Hence, mechanisms are required for efficient selection of appropriate web service instance in terms of quality and performance factors at the time of the web service consumption.

The discovery mechanism should offer a number of capabilities, recognizable at both development and execution time. During development, one may search a web service repository for information about available web services. At execution, client applications may use this repository to discover all instances of a web service that match a given interface in automated way.

The main objective of this research study is to propose a simple mechanism at the level of standards such as WSDL and UDDI which will attempt to select the most efficient web service among possible different alternatives with real-time, optimized and countable factors-parameters. The mechanism aims at minimizing the search of web services by ranking the matched web services based on functional requirements by keyword search and nonfunctional requirement by QoS parameters.

The work aims to examine and analyze the different mechanisms and models for the web service discovery and thereafter attempts to propose the best discovery mechanism for the desired web service.

1.6 Research Methodology

1) The research study approaches the problem defined in section 1.4 in different phases as follows:

   Phase I

   a. Study the existing web service discovery mechanisms to determine their suitability for discovering the most appropriate web service from the available set of services for the desired functionality and find out the suitable ones. A pilot survey study was also conducted whose result signifies necessity of an efficient mechanism which should be able to discover the most appropriate web service as per the consumer's requirement of functionality as well as quality of service (QoS) and the priority of QoS

   b. Identify the QoS parameters for ranking web services for efficient discovery under the given environmental constraints.

   c. Design an algorithm for matching web services with desired functionality based on keyword search and ranking web services based on the QoS parameter values with its preference values specified by the user.

   Phase II

   a. Design an algorithm for calculating monitor ratings and score for each functionally and QoS matched service.

   b. Design an algorithm for ranking the web services based on the overall scores ie. Both actual QoS and monitored QoS score.

   Phase III

   Implement the existing and proposed discovery algorithms designed in Phase I and Phase II and investigate the performance of each based on quality of result obtained.

2) The research work provides

   a. Comparative study of existing web service discovery mechanisms to determine their suitability for discovering the most appropriate web service from the available set of services for the desired functionality and find out the suitable ones
   b. Algorithms for matching and ranking of web services for the selection.
   c. Result analysis of existing discovery algorithm and proposed discovery algorithm based on QoS parameters.

1.7 Organization of Thesis

In Chapter 2, the review of literature and various mechanisms for web service discovery are presented. It starts with describing the concept of web service discovery and further discusses various web service discovery mechanisms. It presents how the search engines like Google, Yahoo are not useful enough for discovering the services available over the internet, as those searches are generic and it could only locate publicly accessible WSDL documents. Various mechanisms to discover web services have been reviewed and presented. It also presents the impact of centralized mechanisms UDDI and ebXML, on the way of conducting the e-business by making it possible for business organizations to publish information on the internet about their products and web services. Decentralized approaches based on Peer-to-peer mechanisms and federated registry are also discussed.

Chapter 3 discusses about services registries available and the data model of each. Two main services registries are discussed namely, Universal Description, Discovery and Integration (UDDI) registry and Electronic Business XML (ebXML) registry. The chapter discusses the architecture and comparative study of both registries.

Chapter 4 discusses the approach of UDDI based mechanisms for web service publishing and discovery in the registry. It presents Reputation-enhanced web service discovery with QoS and Web service QoS-Certifier based web service discovery. The chapter presents introduction of new role in the architecture of UDDI registry – Reputation Manager and Web Service QoS Certifier.

Chapter 5 discusses the new mechanism proposed, i.e. Smart Web Service Discovery enhanced with QoS Monitor, to discover a web service. A detailed discussion on how this mechanism can be implemented in current UDDI registry architecture is presented. The chapter provides various algorithms for publishing web services in this registry, matching, rating and ranking these web services according to service consumer’s functional and QoS request.

Chapter 6 provides results and analysis of various experiments conducted. This chapter provides a comparison of results obtained from experiments with different mechanisms. The chapter provides a framework under which different experiments were conducted and lists the parameters chosen for these experiments. The analysis of the results obtained is also provided.

Chapter 7 presents the summary of research work carried out. Certain claims about contribution to the knowledge made by the research are put forward. This chapter draws conclusions and directions for the further research.

Appendix – I lists relevant definitions for understanding of web service architecture and discovery of web services.

Appendix – II presents an ER Diagram for jUDDI database which stores the information about of web services on the server.

Appendix – III provides the formats of questionnaire required for the pilot study to start with the research.

Appendix – IV contains a copy of all the published papers during this research work.

Chapter 2

Review of Literature

The Web Service Discovery issue in distributed applications has been handled since 2001. The present study demanded a comprehensive understanding of different approaches and mechanism used for discovering web services dynamically. The result of literature survey is presented here.

2.1 Web Service Discovery

Web Service Discovery is “the act of locating a machine-processable description of a web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions”. The goal is to find an appropriate Web service-related resource.[97] Traditionally, the Web service discovery processes involved manual intervention. A set of Web service descriptions are discovered according to user requirements. These service descriptions are manually scanned and those services that satisfy user requirements are selected and composed. In the context of distributed system integration, such manual intervention is unrealistic, cumbersome and time consuming.

The approaches to Web services discovery can be classified as centralized and decentralized. UDDI falls under fully centralized approach that supports replication where central registries are used to store Web service descriptions. Having realized that replicating the UDDI data is not a scalable approach several decentralized approaches have been proposed. Three major operators, namely IBM, Microsoft, and ARIBA provide public UDDI service.

Web service discovery mechanisms include a series of registries, indexes, catalogues, agent based and Peer to Peer-P2P solutions. The most dominating among them is the Universal Description Discovery and Integration-UDDI standard that is currently in version 3.

2.2 Web Service Discovery Mechanisms

Web service discovery mechanisms allow accessing to service repositories and/or “crawling the Web” in the search for services. Since large amount of information is associated with web services, methods to narrow the discovery can be quite complicated and use such semantic information. Search engines such as Google and Yahoo have become a new source for finding Web services. However, search engines do not easily separate and expose to users the basic service properties (i.e. binding information, operations, ports, service endpoints, among others), as they are instrumented or crawling and indexing generic content. In addition, search engines generally crawl Web pages from accessible Web sites while publicly accessible WSDL documents reside on Web servers; hence they are not designed to be fetched and analyzed by normal crawlers.

Web Service Discovery mechanisms are broadly classified into three types :

- Peer-to-Peer mechanisms based on decentralized approach
- UDDI and ebXML registry based mechanisms based on centralized approach
- Alternative mechanisms

2.2.1 Peer-to-Peer mechanisms based on decentralized approach

Peer-to-Peer (P2P) mechanisms are based on decentralized approach in which web services are not discovered on a single registry but it allows web services to be discovered dynamically on the network. from peer-to-peer. All peers in the network are functionally equal and co-operate with each other for responding to the user request. At discovery time, a service requester queries its neighbors in search of a suitable web service. If any one of them matches the request, then it replies. Otherwise each queries its own neighboring peers and the query propagates through the network until a particular hop count or other termination criterion is reached. As peer-to-peer architectures do not need a centralized registry and any node on the network is able respond to the queries it receives, this architectures do not have a single point of failure, such as a centralized registry. Additionally, each peer may contain its own indexing of the existing web services. But at the same time, the reliability provided by the high connectivity of peer-to-peer systems comes with performance costs and no assurance of predicting the path of propagation. Every node in peer-to-peer architecture must have the resources needed to ensure query propagation and response routing. This results that each node acts as a relay of information that may be of no use for the node itself. If the number of nodes on the network are increased , connectivity increases and this results in reducing the efficiency of the system and increasing overhead. Still there may be no guarantee that a request will be propagated across the entire network, and hence there is no guarantee to find the desired web service.

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, *"Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications" 2001* [41]. Chord is a distributed lookup protocol which is designed to efficiently locate the node that stores a particular data item. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Chord is scalable, with communication cost and the state maintained by each node scaling logarithmically with the number of Chord nodes.

Qiang He, Jun Yan, Yun Yang, Ryszard Kowalczyk, Hai Jin, *"Chord4S: A P2P-based Decentralised Service Discovery Approach" 2008* [72] proposes a peer-to-peer based decentralised service discovery approach named Chord4S. Chord4S utilises the data distribution and lookup capabilities of the popular Chord to distribute and discover services in a decentralized manner. Data availability is further improved by distributing service descriptions of functionally-equivalent services to different successor nodes that are organised into a virtual segment in the Chord circle. In addition, the Chord routing protocol is extended to support efficient discovery of multiple services with single request. This enables late negotiation of service level agreements between a service consumer and multiple service providers. They claim that Chord4S achieves higher data availability and provides efficient query with reasonable overhead.

Fatih Emekci, Ozgur D. Sahin, Divyakant Agrawal, Amr El Abbadi*, "A Peer-to-Peer Framework for Web Service Discovery with Ranking" 2004* [19].They have proposed a structured peer-to-peer framework for web service discovery in which Web services are located based on both service functionality and process behavior. It represents the process behavior of the web services with finite automata and use these automata for publishing and querying the web services within the system. The model is scalable and robust due to the underlying peer-to-peer architecture. Web services can join and leave the system dynamically. We also propose an efficient and scalable reputation model based on sketch theory. Thus the returned services are ranked based on the trust and quality ratings of the services using the proposed reputation model.

Ioan Toma, Brahmananda Sapkota, James Scicluna, Juan Miguel Gomez, Dumitru Roman, and Dieter Fensel*, "A P2P Discovery mechanism for Web Service Execution Environment" 2005* [40]. They have presented a scalable approach for automatic discovery of services over distributed execution environments. The solution is based on P2P technologies that proved to be scalable, efficient and robust solutions for distributed systems. As shown in Fig. 2.1, equal WSMX peers which participate in the service discovery process have to match the local registered services against a broadcasted query. A major aspect that is to be considered in this context is the topology of network. For message routing the topology of the network has significant impact on the overall performance of the service discovery process. The approach that they have adopted to address these aspects is the HyperCuP approach. HyperCuP decreases the big overhead of network communication by providing a topology based on a structure called hypercube: a generalization of a 3-cube to n dimensions. In the resultant graph the connection between neighbored nodes can be associated with a specific dimension of the hypercube. This allows us to define a message broadcast scheme with certain guarantees: nodes receive a message exactly once and the number of messages sent is linearly dependent on the number of nodes in the network. A set of structuring ontology concepts is used to build a hypercube consisting of distinct concept clusters. A query which consists of a logical combination of service and domain ontology concepts is routed to all relevant concept clusters. Within a concept cluster the message is broadcasted to all contained peers. If the query formulation matches the conceptual description of a service the representing peer is reacting by sending an according response to the requester.

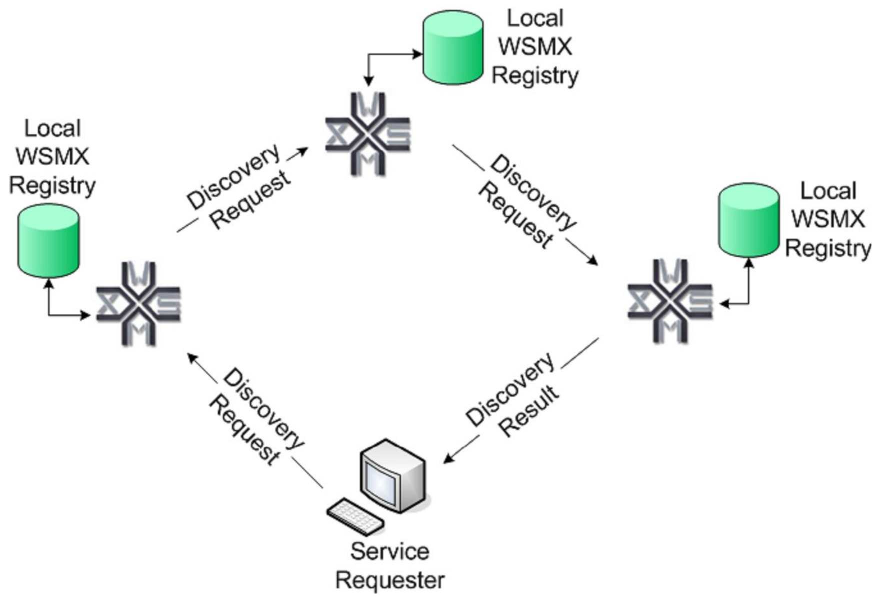


Figure 2.1 Peer-to-Peer approach for Distributed Discovery in WSMX

Farnoush Banaei-Kashani, Ching-Chien Chen, and Cyrus Shahabi, *"WSPDS: Web Services Peer-to-peer Discovery Service" 2004* [22]. They have introduced WSPDS (Web Services Peer-to-peer Discovery Service), a fully decentralized and interoperable discovery service with semantic-level matching capability. They claim that a peer-to-peer architecture of the semantic-enabled WSPDS not only satisfies the design requirements for efficient and accurate discovery in distributed environments, but also is compatible with the nature of the web Services environment as a self-organized federations of peer service-providers without any particular sponsor. WSPDS is a distributed discovery service implemented as a cooperative service. A network of WSPDS servants collaborate to resolve discovery queries raised by their peers. Fig. 2.2 depicts an unstructured peer-to-peer network of WSPDS servants. Each servant is composed of two engines, communication engine and local query engine, playing two roles: (1) Communication and Collaboration : the communication engine provides the interface to user and also represents the servant in the peer-to-peer network of servants. This engine is responsible for receiving service queries from users, resolving the queries by local query (through the local query engine) and global query (via its peer servants), and finally merging the received responses to reply to the user query; and receiving queries from its neighbors in the peer-to-peer network, resolving the queries by local query, and sending the response (if not empty) to the network as well as forwarding the query to other neighbors in the network. (2) Local query: the local query engine receives the queries from the communication engine, queries the local site (where the servant is running) for matching services, and sends responses to the communication engine.

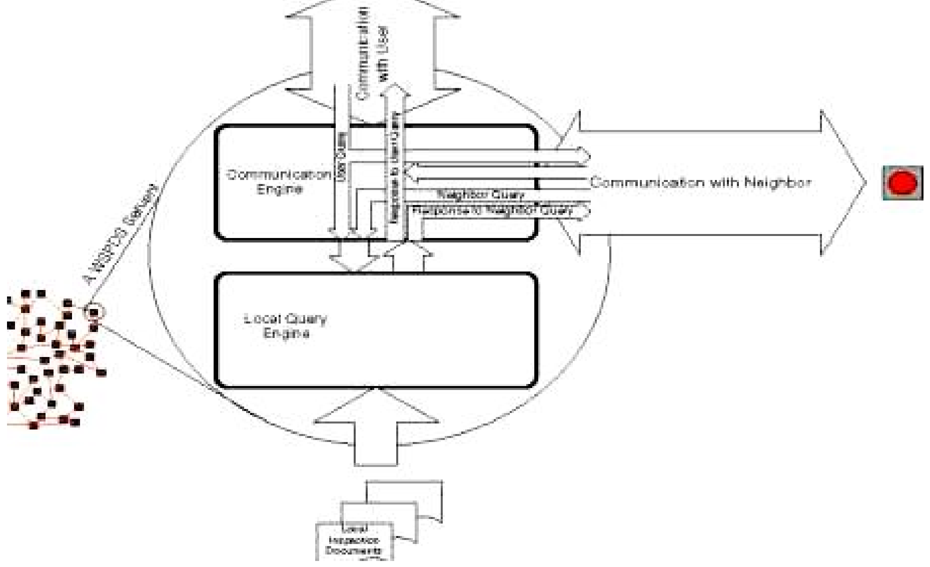


Figure 2.2 WSPDS Architecture

Sivashanmugam, K., Verma, K., Mulye, R., Zhong, Z., and Sheth, A., *"Speed-R: Semantic P2Penvironment for diverse Web Service registries" 2004* [95]. They have proposed Speed-R system for publishing and discovering web services that uses ontologies and a P2P infrastructure. Some nodes in the P2P subsystem are assigned registries, which in turn partitioned according to their specific domain. An ontology is assigned to each domain. Its architecture is based on role assignment to peers. e.g. some nodes have undertaken the role of controlling updates and propagating them, thus their system may suffer from single point failure. Fig. 2.3 presents architecture of Speed-R system. Each Peer runs 'Operator Peer' to control semantic access to its registry (direct registry access without support for semantic discovery is allowed). Peers support Domain Ontology and Operator Services (if ontology is not used, no semantic discovery can be provided, search defaults to keyword search). Each Registry can be accessed using API, which is dependent on its implementation and standard that it conforms to. Registries Ontology (i.e., the upper ontology, only one for the whole P2P cloud) is present in the P2P network. Any given time peers are aware of the updated Registries Ontology.

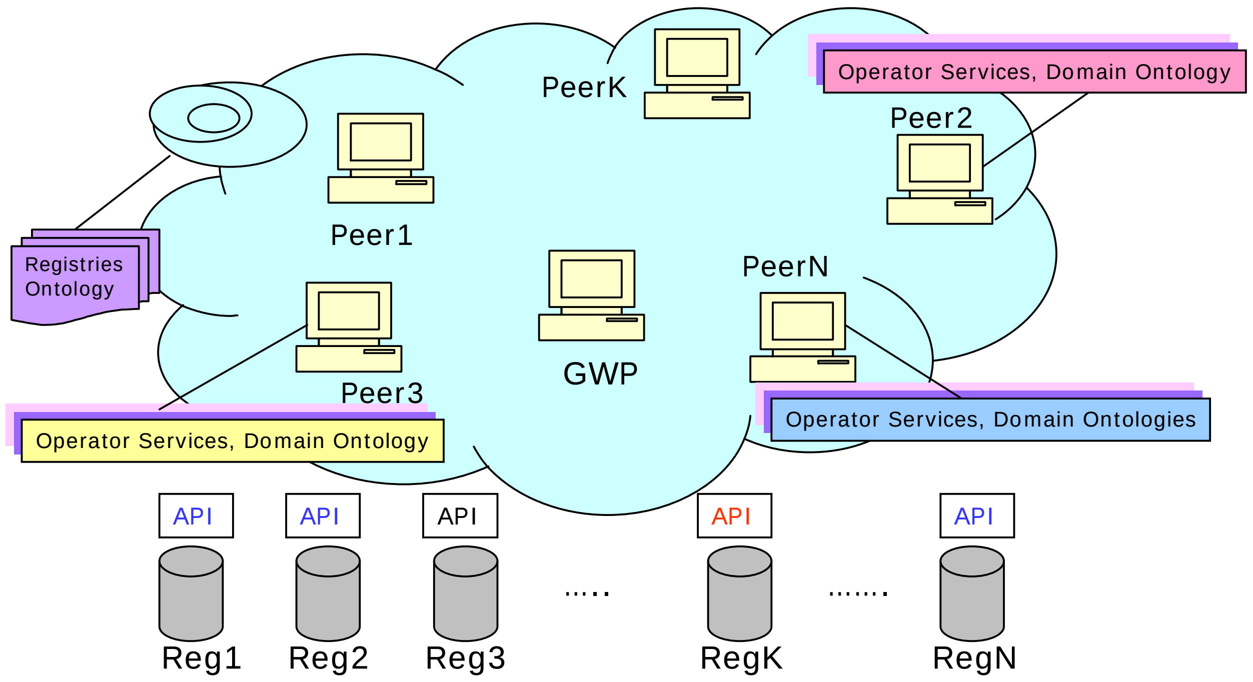

PeerK
Operator Services, Domain Ontology
Peer2
Peer1
Registries Ontology
PeerN
GWP
Peer3
Operator Services, Domain Ontology
Operator Services, Domain Ontologies
API
API
API
API
API
.....
....
Reg1
Reg2
Reg3
RegK
RegN

Figure 2.3 Speed-R Architecture

Gang Zhou, Jianjun Yu, Rui Chen, Hui Zhang, *"Scalable Web Service Discovery on P2P Overlay Network" 2007* [25]. They have developed the ServiceIndex system for service discovery which merges advantages of P2P computing and Semantic Web Services into web services world. The ServiceIndex system tries to solve the problem of semantic search in distributed environment and support complex search, tree lookups, locality sensitivity, and ontology based service discovery. It is possible to construct a dynamic and pure P2P overlay network for service discovery and achieve considerable system performance.

Summarizing the above papers, it can be concluded that peer-to-peer discovery mechanism provide an efficient and scalable solution for the discovery of services in distributed systems as it is fully decentralized and do not have a single point of failure, such as a centralized registry. Additionally, each peer may contain its own indexing of the existing web services. In case of semantic search, ontology can also be specified.

2.2.2 UDDI and ebXML registry based mechanisms based on centralized approach

In centralized approach, UDDI (Universal Description, Discovery and Integration) and ebXML (electronic business XML) are the two types of registries which are storing and managing web service information centrally. UDDI is a vendor-sponsored initiative led by IBM, Microsoft, and Ariba, whereas, ebXML is a UN/CEFACT (United Nations center for Trade Facilitation and Electronic Business) / OASIS sponsored initiative for creating a single global electronic market. UDDI and ebXML, make it possible for business organizations to publish information on the Internet about their products and web services, where the information can be readily and globally accessed by clients who want to do business. UDDI Registry is a web-based registry that exposes information about a business providing web service, web service and its technical interfaces. A service provider makes its services available to public users by publishing information about the service in a UDDI registry.

The information about Web services in a UDDI registry includes a description of the business and organizations that provide the services, a description of a service's business function, and a description of the technical interfaces to access and manage those services [92]. A UDDI registry which is an XML- based registry consists of instances of four core data structures including the businessEntity, the businessService, the bindingTemplate and the tModel. This information comprises everything a user needs to know to use a particular Web service. The businessService is a description of a service's business function, businessEntity describes the information about the organization that published the service, bindingTemplate describes the service's technical details, including a reference to the service's programmatic interface or API, and tModel defines various other attributes or metadata such as taxonomy and digital signatures [92]. UDDI (Universal Description, Discovery and Integration) plays a key role in the web service architecture. It provides a structured and standard description of the web service functionalities as well as searching facilities to help in finding the providers that better fit client requirements. Generally speaking, a UDDI registry contains the information about businesses and services these business organization offers. These services may not be always web services or computer related services at all. In fact, UDDI was designed in the intention of holding arbitrary information about a business. It serves not only as an access point for service related information, but also about the businesses themselves. Structure of UDDI is similar to telephone directory, in the way that phone numbers are stored and catalogued.

With ebXML, companies are able to define how to conduct business using a specific vocabulary. Core components are used to build predefined documents. Messages are sent using standardized protocols and formats. All of this information is stored in ebXML registries. Business Processes and Business Document has to be created prior to their use. Specification of these both describes the workflow of business processes and the information exchanged between the partners respectively. These documents can be composed of reusable and extendable Core Components. An ebXML Registry provides means for finding organizations, business processes, core components and other objects. Therefore it does not store the actual objects but metadata and associations between them. Business partners register their services in an ebXML registry along with their Collaboration Protocol Profiles (CPPs). During the search the registry is queried for a business partner that offers the required service. Based on the CPPs of both partners a Collaboration Protocol Agreement (CPA) is formed which specifies what kind of business is to be performed and how. Usually CPA is negotiated after being proposed by one party. Based on the agreement it is now possible to configure an ebXML enabled application and execute the business process.

Ali ShaikhAli, Omer F. Rana, Rashid Al-Ali, David W. Walker, “*UDDIe: An Extended Registry for Web Services*” 2004 [79]. They implement UDDIe an extension to UDDI, which supports the notion of “blue pages”, to record user defined properties associated with a service and to enable discovery of services based on these. UDDIe enables a registry to be more dynamic, by allowing services to hold a lease – a time period describing how long a service description should remain in the registry. UDDIe can co-exist with existing UDDI – and has been implemented as an opensource software.

Extensions in UDDIe are based on four types of information: business information; service information, binding information; and information about specifications for services. A service may be discovered by sending requests based on service information. The extensions provided in UDDIe consist of the following:

Service Leasing: Service providers may want to make their service available for limited time periods (for security reasons, for instance) – or the service may change often. UDDIe supports "Finite" and "Infinite" leases – where a finite lease can be immediate, or based on a future lease. When using finite leases, service providers must define the exact period for which the service should be made available for discovery in the registry. The lease period is restricted by the maximum allowable lease period defined by the UDDIe administrator. Depending on the type of application domain for which the UDDIe registry is to be used, the value of the maximum allowable lease may change. This parameter is left to the UDDIe administrator to set. For example, if a service provider is interested in publishing a service in UDDIe for two hours, but the maximum granted lease is one hour, publication of the service will be rejected by the registry. A "future lease" allows a service provider to make the lease period start at a future time – the service will only be discoverable once this lease has been activated. Alternatively, service providers may want to publish their services for an infinite period of time. Such leases are allowed in UDDIe, but only if the ratio of finite/infinite lease services is within a threshold (a parameter set by the UDDIe administrator).

Replication: The UDDI Business Registry (UBR) is conceptually a single system built from a group of nodes that have their data synchronized through replication.
A series of operator nodes each host a copy of the content, thereby replicating content among one another. Content may be added to the UBR at a single node, and that operator node becomes the content master. Any subsequent updates or deletes of the data must occur at the operator node where the data was inserted. UDDIe can be used as a private operator node that is not part of the UBR. Private nodes do not have data synchronized with the UBR, so the information contained within is distinct. The availability of private nodes is significant if an organization considers sharing

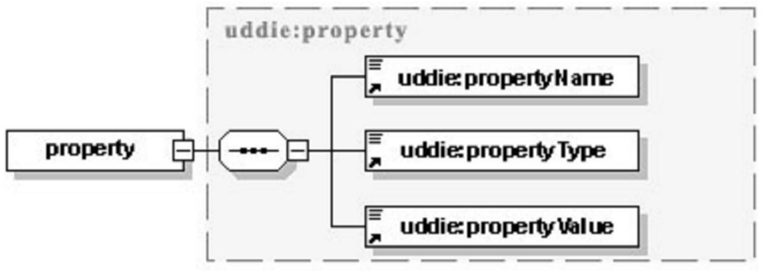


Figure 2.4 Property attributes

their service content a security problem. This is useful in instances where a company does not want to expose certain service offerings and business processes to others – for instance, suppliers set up to handle large contracts may not be able to handle individual customers.

In UDDIe a business Service, structure represents a logical service – and is the logical child of a business Entity – the provider of the service. Service properties are contained in the property Bag entities – such as the Quality of Service (QoS) that a service can provide, or the methods available within a service that can be called by other services. Figure 2.4 illustrates the attributes associated with a property – and consists of a propertyName, propertyType and propertyvalue. Some of these are user defined attributes – such as propertyType – and can be number, string, method etc. Range based checks, for instance, are only allowed if the propertyType is a number. The API for interacting with the registry system extends three classes within existing UDDI implementations. The extensions provided in the API include: _ saveService: This set of APIs is mainly used for publishing service details. This has been extended from the original UDDI system to introduce dynamic metadata for services. Such metadata could be used to represent attributes such as cost of access, performance characteristics, or usage index associated with a service, along with information related to how a service is to be accessed, and what parameters the service will return. The saveService call utilises the propertyBag mechanism provided in UDDIe. _ findService: This set of APIs is

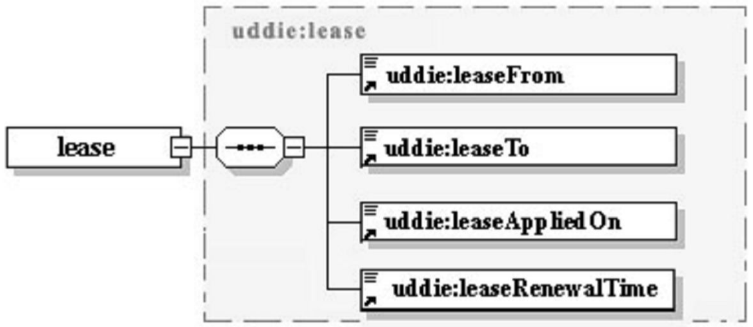


Figure 2.5 The “Lease” element

mainly used for inquiry purposes. In particular we extend this set of API from the original UDDI to include queries based on various information associated with services, such as Service Property and Service leasing.

They claim that extensions to the UDDI registry and query mechanisms would add a great search flexibility, making UDDI a more powerful search engine. The ability for UDDIe to co-exist with standard UDDI version is also an important aspect of this work – as they do not break compatibility with existing UDDI deployments.

Phil Bonderud, Sam Chung, Barbara Endicott-Popovsky, *"Toward Trustworthy Service Consumers and Producers" 2008* [11]. They proposed a S-QoS4WS approach that utilizes 'PublisherAssertion' tags within the UDDI to satisfy Security and QoS issues. This approach makes use of existing mechanisms within UDDI version 3 to resolve current issues involving trust and non-repudiation. S-QoS4WS takes into consideration security and QoS issues with respect to establishing trust and nonrepudiation. The approach adds an optional third party entity to the web services paradigm whose sole purpose is to certify information about each respective business partner. The third party service certifier certifies that services offered by a service producer meet the specifications used to describe the service in the UDDI. The third party consumer validation entity authenticates that its service consumer partner is a trustworthy and legitimate business. Each third party entity is expected to publish its own web service whose sole purpose is to provide an automated way of obtaining information.
In Figure 2.6, solid lines represent interactions that require human intervention. It is expected that in order for a service to be adequately certified or a consumer to be validated, that some degree of human involvement will be required. Dashed lines represent transactions that are fully automated. S-QoS requires that a service producer select a third party entity (A) which will certify that any statistics and requirements it wishes to advertise in the UDDI, about a service, are accurate. This communication is expected to require human involvement, which is indicated by the solid line in Figure. Upon reaching final agreement concerning a service's certification (B), the certifying entity publishes a web service to a UDDI. This service, published by the certifier, holds the results of a service's certification. The service producer (B)

also publishes its service to the UDDI, if it has not already done so. Both the service producer and the third party certification entity make identical ‘publisherAssertions’ for this service,

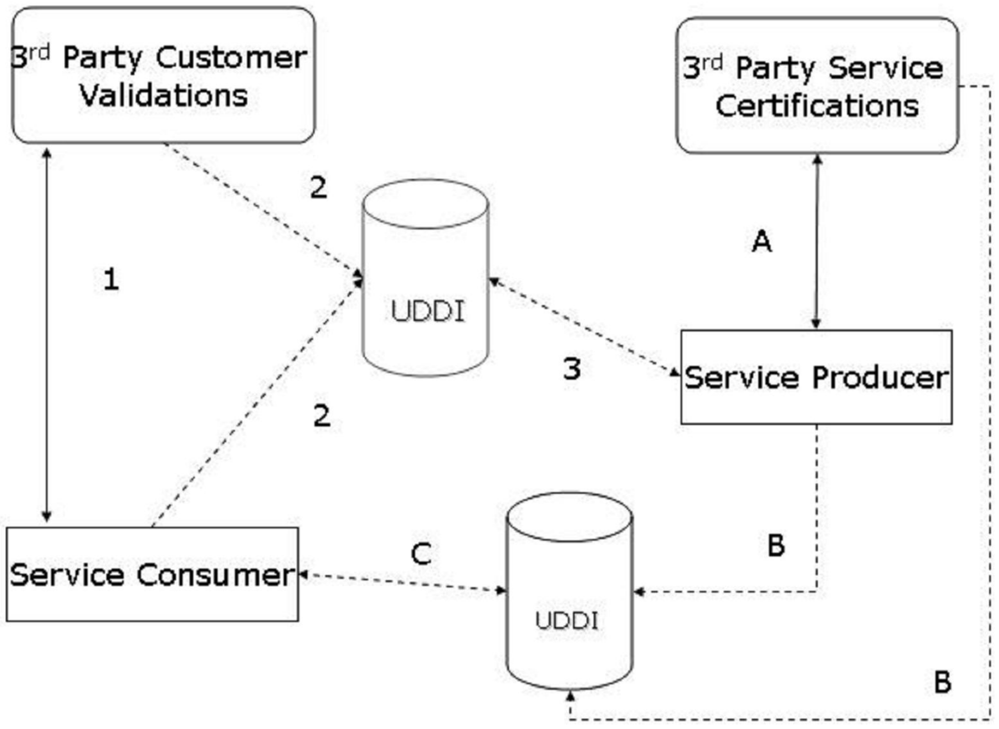

3rd Party Customer Validations
3rd Party Service Certifications
2
1
UDDI
A
3
Service Producer
2
Service Consumer
C
B
UDDI
B

Figure 2.6 : Service model of UDDI

which will be explained in detail in the next section. By making identical ‘publisherAssertions’ for this service (C), service consumers can query the UDDI for ‘status:complete’ certified services. S-QoS mirrors the interactions between the service producer and its certification entity to produce consumer validations (1 – 3). Whether or not a unique UDDI is used as diagramed, which caters only to service consumers, is irrelevant to this research and not a requirement for the success of this approach. Equivalent to communications represented by line A, communications between a service consumer (1) and its respective third party service consumer validation entity is expected to require human involvement. Upon reaching final agreement over the information to be published (2), the validation entity publishes a web service to a UDDI that holds the results of a consumer’s validation. The service consumer (2) also publishes an informational service to the UDDI that represents itself, if it has not already done so. Both the service consumer and the third party validation entity make identical ‘publisherAssertions’ for the consumer. By making identical ‘publisherAssertions’ for the consumer (3), service producers obtain an added bonus of being able to query a UDDI for 'status:complete' validated consumers. This added bonus enables service producers to proactively market their services to viable organizations. Within each UDDI businesses have the option of providing a service description statement. Service consumers will enter 'Service Consumer' as their descriptor, consumer evaluators will enter 'Consumer Validation', service certification entities will use 'Service Certification', and service producers default to any description.

***Youngkon Lee* "*Web Services Registry implementation for Processing Quality of Service*"** *2008* [103]. This paper presented the design principle for integrating quality management on Web service registry developed in UDDI specification and Web service quality management system (WSQMS). WSQMS, developed by NIA1 can measure and collect the quality information of Web services by its agency system installed on the Web service system. Web service registry is core system for registering and searching WSDL(Web Service Description Language). In a Web service registry, *WSDL* is referenced in a *tModel,* which is a container for a reference to the *WSDL*. Because *tModel* is devised to include the detail information about a Web service, it is natural conclusion that we modify *tModel* to be proper for including the reference to *WSQDL*. There are two choices. First is to make a new reference data object to *WSQDL* in <*overviewDoc*> as the form described as *WSDL*. This way is trivial, so it enables

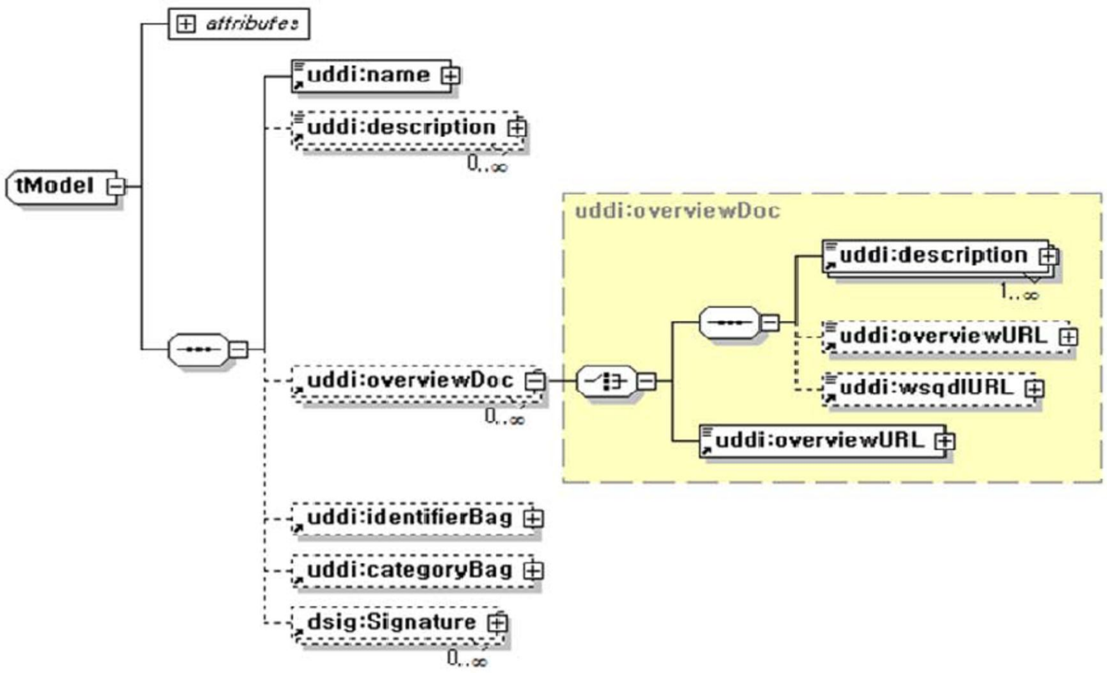


Figure 2.7 XML schema for *WSQDL* complex type.

users to find out at once that *WSDL* and *WSQDL* describe characteristics for the same target Web service. This way, however, restricts severely the usage of reference to quality data. That is, user cannot search the quality data rapidly because there are no classification schemes for quality data. Figure 2.7 shows <*wsqrlURL*> in <*overviewDoc*> and XML schema for *WSQDL* complex type.

Second way is to make a specific tag, <*qualityBag*>, in *tModel* to store the reference to *WSQDL*. This requires additional processing modules, but enables the quality data to be used more widely. For example, this method allows the reference to *WSQDL* in <*qualityBag*> to be handled as the form of *tModel*, resulting that process related with *tModel* could have still flexibility. However, it requires updates of considerable part of the registry because the registry system should process two types of *tModel* for: *WSDL* and *WSQDL*. However, it is impossible to search a Web service effectively on the basis of quality data, because *tModel* has only reference data to *WSQDL*. Thus, it is desirable to implement architecture for referring Web service quality data by using the quality classification scheme. Figure 2.7 shows the *tModel* component structure and XML schema including <*uddi:qualityBag*>.

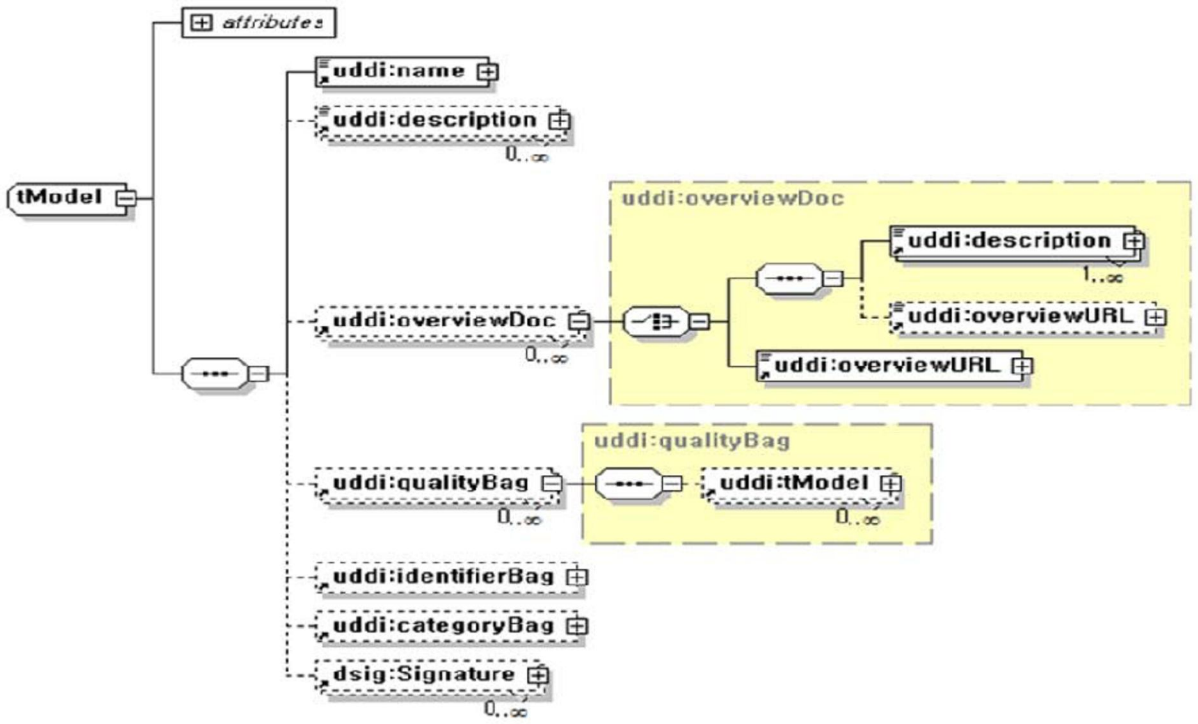


Figure 2.8 *tModel* Component and Schema including qualityBag

In Figure 2.8 the structure of <*overviewDoc*> is the same as previous *tModel*, but <*qualityBag*> is a new structure for referring any number of *tModel*. Another way is to add quality context information to *tModel* for quality classification scheme. This allows the registry to have quality context in <*qualityBag*>, as corresponding *WSDL* through *tModel*. As the previous search method of Web service registry by using <*categoryBag*>, a registry parses previously the quality data in <*qualityBag*> and stores the quality context so that users may just search a Web service satisfying some criteria by using the quality context or in quality classification. To represent quality context data consistently and to manage it requires further study. Figure 2.8 shows the <*qualityBag*> component structure and its XML schema including <*qualityContext*>, whose structure could include any type of character string. <*qualityBag*> stores any number of required <*qualityContext*> and represent any type of quality data. For example, as digital signature for message consistency and proof of message sender, a <*qualityContext*> as type of /eval/sec/Dsig/keySize/ could be made and we say that a system is safer when it has its value of 128 rather than 64 in the respect of digital signature safety. <*qualityContext*> representing Web service quality information should be registered on a registry and user can search the quality data according to the value of <*qualityContext*>. The registry requires the additional APIs for processing the quality data in the relationship with *WSQMS*. Firstly, it is required for *WSQMS* to have APIs searching the new registered Web service. The APIs correspond to the functionality of searching Business Entity, Service, Binding, and *tModel*. APIs for representing the reference to the quality information sent from *WSQMS* are required. If the reference to the quality data is stored in *tModel*, the additional APIs for processing *tModel* operation are required. Besides, it's required the APIs for modifying and updating Web service quality information and synchronizing the Web service information between *WSQMS* and registries.

Massimo Paolucci and Katia Sycara, "Autonomous Semantic Web Services" 2003[66]**.** In this paper, the authors presented a mechanism that begins to bridge the gap between the Web services infrastructure and the Semantic Web. They adopted the vision of Web services as autonomous goal-directed agents that select other agents to interact with and that flexibly negotiate their interaction models, acting variously in client–server and peer-to-peer modes. The resulting web services called as *autonomous Semantic Web services*, use ontologies and semantically annotated web pages to automate the fulfillment of tasks and transactions. In particular, these services use the Semantic Web to support capability- based discovery and interoperation at runtime. A first step toward this vision was to develop formal languages and inference mechanisms for representing and reasoning with core Web service concepts. The DARPA Agent Markup Language for Services (DAML-S) is the first attempt to define such a language. One objective behind the Semantic Web is to provide languages for expressing the content of Web pages and making that information accessible to agents and computer programs. More precisely, the Semantic Web is based on a set of languages such as the Resource Description Framework (RDF), DAML+OIL, and the more recent Web Ontology Language (OWL), which can be used to annotate Web content. These languages have well-defined semantics and inferential procedures that let agents draw inferences from the languages' statements. Using the semantic markup for the US National Oceanic and Atmospheric Administration's page reporting Pittsburgh's weather conditions, for example, an agent could learn that the current condition is heavy snow. The agent might further learn from the Pittsburgh school board site's semantic markup that all schools are closed on days of heavy snow. Combining the two pieces of information, the agent could infer that Pittsburgh schools are closed today. The Semantic Web's second element is a set of ontologies that provide conceptual models for interpreting the information provided. An ontology of weather might contain concepts such as temperature, snowy, cloudy, and sunny, for example, and relationships between the terms. The Semantic Web vision is about transforming the Web into an Internet-wide knowledge-representation system in which ontologies provide the conceptual framework for interpreting the information provided by Web pages. To produce the types of inferences they have described, the Semantic Web requires computational processes and agents that can interpret semantic content and derive consequences from the information they collect. The Semantic Web also supports a more distributed computational model in which a requester transacts with multiple Web services, solving problems through collaboration and negotiation. Within this scheme, ontologies not only define a shared conceptualization for interpreting semantic markup of Web sites, but also provide a shared vocabulary that lets services across the Web use the same terminology to interpret each other's messages. Ultimately, the Semantic Web will provide the basic mechanisms for extracting information from Web pages and the basic knowledge that Web services will use in all transactions. In addition to knowledge, however, Web services need an infrastructure that facilitates reliable communication — registries to locate other services, reputation services, guarantees of secure and private transactions, and so on. Such an infrastructure falls outside the current view of the Semantic Web's scope.

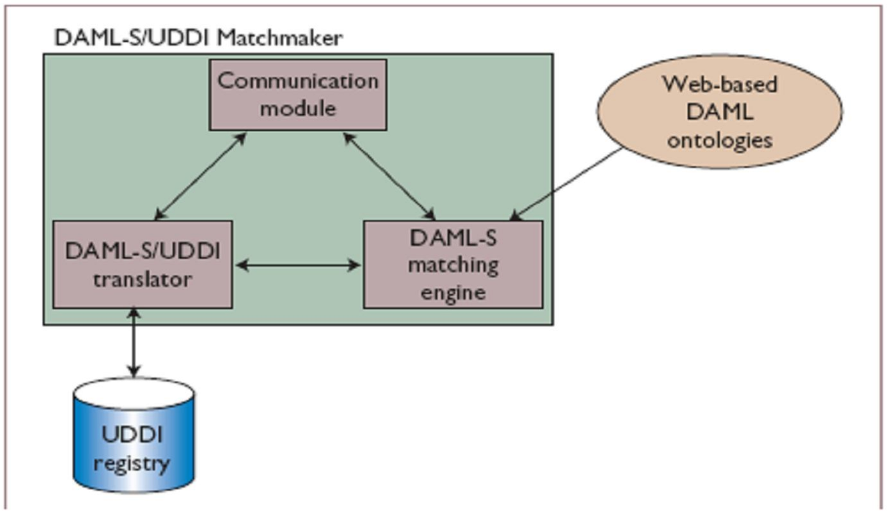


Figure 2.9 DAML-S/UDDI Matchmaker architecture

Web services advertise or request through the communication module using DAML-S. Advertisements are stored in the UDDI registry, and requests are sent to the DAML-S matching engine. The *service profile* provides a high-level view of a given Web service. It is the DAML-S analog to the Web service representation that UDDI provides in the Web services infrastructure, although the two have some sharp differences as well as similarities. Some information, such as a Web service's provider, is present in both descriptions, but the service profile supports properties such as the representation of capabilities — the tasks the service performs — that UDDI does not support. On the other hand, UDDI describes the ports the Web service uses, whereas DAML-S relegates this information to other modules of the description, such as the grounding (described below). The *process model* specifies the tasks a Web service performs, the order in which it performs them, and the consequences of each. A client can use the process model to derive the service's choreography, or message-exchange pattern, by figuring out what inputs it expects, when it expects them, what outputs it reports, and when. The process model's role is similar to emerging standards such as BPEL4WS and WSCI, but focuses more on the effects of executing a service's different components. The *service* *grounding* binds the abstract description of a Web service's information exchanges —defined in terms of inputs and outputs in the process model — with an explicit WSDL operation, and through WSDL to SOAP messages and transportlayer information. DAML-S's reliance on DAML+OIL, as well as WSDL and SOAP, shows how proposed Web services standards can be enriched with semantic information. DAML-S adds formal content representations and reasoning about interactions and capabilities to Web service specifications. Therefore, DAML-Senabled Web services use UDDI, WSDL, and SOAP to discover other services and interact with them, and they use DAML-S to integrate these interactions, in their own problem solving.

Managing Web Services with DAML-S

They have implemented tools for Semantic Web service discovery and invocation making use of DAML-S and complementing current Web services systems. They describe the DAML-S/UDDI Matchmaker and the architecture of a DAML-S-empowered Web service.

DAML-S-Enabled Service Discovery

The DAML-S service profile relies on ontologies to specify what type of information the Web service reports and what effects its execution produces. At discovery time, a Web service generates a request that contains a profile for the ideal service it wants to interact with. The discovery process selects a Web service provider's profile that matches the request. Although DAML-S profiles and UDDI Web-service descriptions contain different information, they share the goal of facilitating Web-service discovery. The combination could thus provide rich representations for Web services. Using UDDI's TModels to encode DAML-S capability descriptions, we can reconcile the differences between the two. Once the capabilities encoded, a new module is added to UDDI: the *matching engine* performs inferences based on DAML+OIL logics and effectively adds capability matches to UDDI. The result is the DAML-S/UDDI Matchmaker for Web services. The Matchmaker receives Web-service advertisements, information inquiries, and requests for capabilities through the *communication module*, which implements a simple inquiry-and-publish API. The communication module then sends the advertisements and inquiries to UDDI through the *DAML-S/UDDI translator*, which transforms DAML-S encoded advertisements into UDDI format. The communication module directs requests for capabilities to the DAML-S matching engine, which selects those Web services whose advertised capabilities match the request. The matching is complicated by the fact that providers and requesters have different views on Web-service functionality. Thus, the matching engine can't base the selection on strings or keywords. Rather, it must match semantic descriptions of capabilities to access the deeper meaning of the advertisements and requests. Consider, for example, a service provider advertising that it sells pet food, and a requester looking to buy dog food. A UDDI-style registry would be unable to match the request because keyword matching is not powerful enough to identify the relationship between pet food and dog food. Instead, DAML-S profiles let service providers express concepts that are explicitly related via ontologies. In this case, the provider could specify that dog is a type of pet, and the DAML-S matching engine could recognize a semantic match between the request and the advertisement. The DAML-S matching algorithm accommodates the differences between an advertisement and a request by producing flexible matches — recognizing degrees of similarity — on the basis of available ontologies. Basically, the matching engine attempts to verify whether the requested outputs are a subset of those generated by the advertisement, and whether the advertisement's inputs subsume those of the request. When these conditions are satisfied, the advertised service generates the outputs that the requester expects and the requester can provide all the inputs the Web service expects. The degree of satisfaction between these two rules determines the degree of match between provider and requester.

Katia Sycara, Massimo Paolucci, Julien Soudry, and Naveen Srinivasan, *"Dynamic Discovery and Coordination of Agent-Based Semantic Web Services" 2004* [87]. Matchmaking and brokering are multiagent coordination mechanisms for Web services. Both have performance trade-offs, but the Web Ontology Language for Semantic Web Services (OWL-S) can handle extensions that address some of the shortcomings. In this article, the authors focus on the broker, analyzing both its interaction protocol and reasoning tasks. The authors also describe OWL-S's exec extensions, detail their implementation's basic features, and explain how these features address the broker's reasoning problems.

M. Adel Serhani, Rachida Dssouli, Abdelhakim Hafid, Houari Sahraoui, *"A QoS broker based architecture for efficient web services selection*" 2005 [78]. In this paper, the authors presented a QoS broker based architecture for web services. The main goal of the architecture was to support the client in selecting web services based on his/her required QoS. To achieve this goal, researchers proposed a two-phase verification technique that is performed by a third party broker.

The first phase consists of syntactic and semantic verification of the service interface description including the QoS parameters description. The second phase consists of applying a measurement technique to compute the QoS metrics stated in the service interface and compares their values with the claimed one. This is used to verify the conformity of a web service from the QoS point of view (QoS testing). A methodological approach to generate QoS test cases, as input to QoS verification is used. They implemented a prototype that included the verification and certification components of the broker. They performed experiments to evaluate the importance of verification and certification features in the selection process using real web services. The architecture extends the standard Service Oriented Architecture (SOA) [1] [2] with QoS support for web services. It includes QoS description during the service publication, and performed dynamic QoSaware invocations. In addition, it verified, certified, confirmed and monitored QoS dynamically via a web service-based broker. The architecture involves four main participating roles the web service broker, the web service provider, the

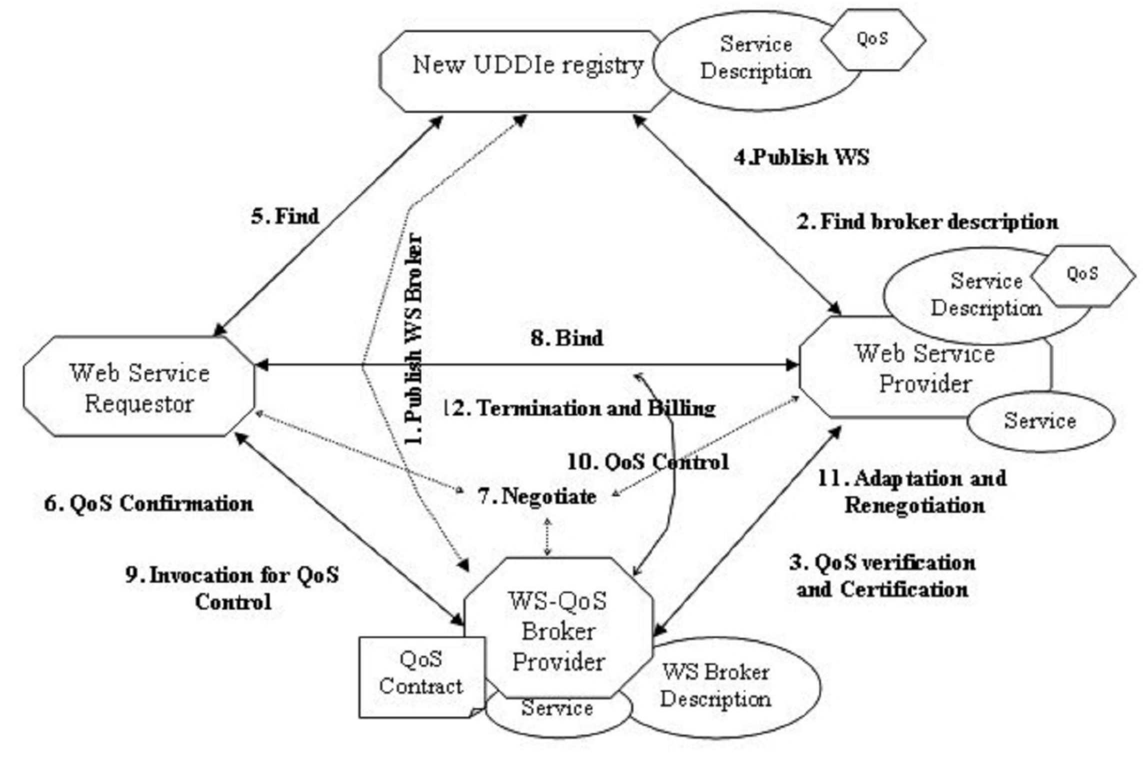


Figure 2.10 QoS broker based architecture

client, in addition to a QoS enabled UDDIe registry [15].

Figure 2.10 presents an architecture based broker with features such as support of service selection based on client requirement, QoS verification and certification. QoS verification is the process of validating the correctness of information described in the service interface as well as the described QoS parameters. The QoS verification is performed using an approach that generates test cases to measure QoS parameters. The verification will be used as input for the certification process that will be issued when the verification succeed. The broker arbitrates the negotiation process between clients and their providers until they reach an agreement. During web service invocation, the broker measures dynamically QoS attributes and uses their values to monitor the provision of the selected QoS level; then, it notifies the interested entities of any violation. The broker updates, regularly, its database whenever significant changes happen. In the architecture, the certification process goes beyond certifying just the QoS provider's claims.

Wenli Dong "QoS *Driven Service Discovery Method Based on Extended UDDI*", 2007 [99]. In this paper the author proposed a QoS driven service discovery method based on extended UDDI with the help of Semantic Web. First, a Extending UDDI Model based on QoS driven was proposed, QoS ontology was analyzed to reduce misunderstanding. Second, a matching algorithm based on fuzzy correlation calculate was proposed to filter the unqualified service to improve the discovery accuracy. Third, a discovery process based on policy was built based on Semantic Web technology. The experience results showed that the QoS driven Web service discovery method possessed high discovery accuracy.

The QoS certifier was added in the proposed extended UDDI model to support QoS filtering function as shown in Figure 2.11. The QoS certifier's role is to verify service provider's QoS claims. According to the author the proposed registry differed from the current UDDI model by having information of the function description of the Web service as well as its associated quality of service registered in the registry repository. Lookup could be made by function description of the desired Web service, with the required quality of service attributes as lookup constraints. QoS is a combination of several non-functional characteristics. QoS publication helps selecting among services with the same functionality based on OoS. There are many aspects of QoS that are important to Web services.

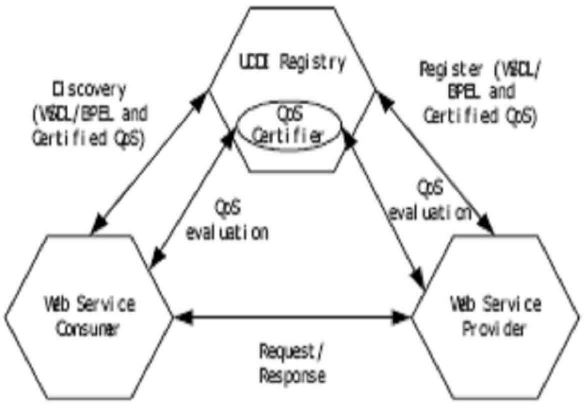


Figure 2.11 Architecture of Web Service Publication and Discovery with QoS certifier

Huimin HE, Haiyan DU, Dongxia HAN, Yuemei HE, ***"****Research on the Models to Customize Private UDDI Registry Query Results" 2008* [37] . This paper presents three models which enable the customization of Universal Description, Discovery and Integration (UDDI) query results, based on some pre-defined and/or real-time changing parameters. These proposed models detail the requirements, design and techniques which make ranking of Web service discovery results from a service registry possible. They present an extension to the UDDI inquiry capabilities to customize or rank the query results, based on business requirements. Authors proposes three models to achieve the customization of UDDI query results. All three share some common architecture components as shown in Fig.2.12.

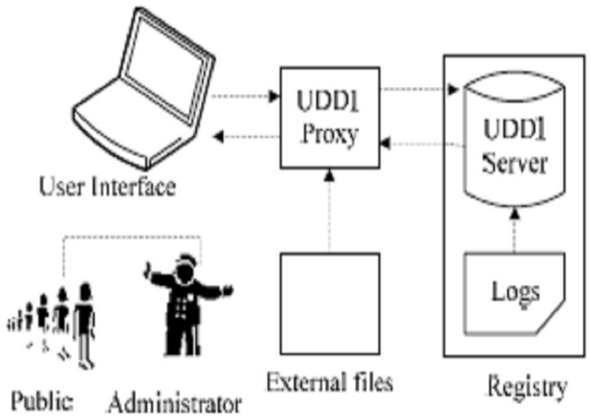


Fig 2.12 Common Architecture components of the Models to Customize Private UDDI Registry Query Results

They are: UDDI server, UDDI Proxy and User Interface. These components will interact with other external components. The customization criteria required is the ranking of list of business or service list to User Interface. Load balancing also can be improved by keeping the User Interface and UDDI Proxy on separate servers. The most basic feature of UDDI is to allow businesses to publish their services in a directory and enable other business representatives to locate partners and to form business relationships based on the web services they provide. They introduce two types of parameter: static and dynamic. The static parameter will hold certain values which has been fixed and do not change during run-time. Only Administrator access can modify its values. Examples of static parameter are vendor ranking, cost per transaction and advertisement priority. Unlike static, the dynamic parameter will be used to store value which is real-time changing and gets updated during run-time. The updating frequency will depend on mechanism defined within the criteria. One usage of dynamic parameter is to keep track of service or business popularity. The criteria used to customize the UDDI query results will be represented by static and dynamic parameters.

Model where parameters are saved and retrieved from UDDI server

In this first model, we propose the use of only UDDI Proxy and UDDI Server components, where the parameters will be saved inside the UDDI server itself.

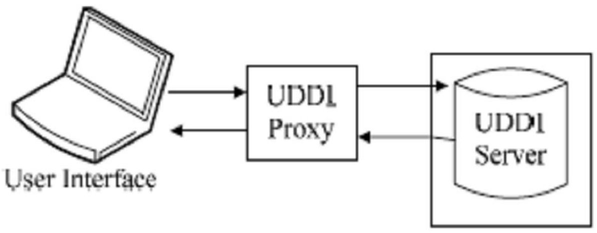


Figure 2.13 Model 1 - Parameter values to be saved and retrieved from UDDI server

This will require a new tModel definition to describe the parameters information. Each business entity and service will then contain a reference to this tModel in their record. The term “bag” indicates a generic container of multiple values, and enables a company to register multiple business identifiers. i. Retrieving Parameters Values In this model, all the parameter values are stored using XML schema inside the UDDI server. Whenever a request is made by consumer to get a list of services, the UDDI Proxy will invoke the UDDI Find functions of the inquiry API. Certain Find Qualifiers can also be used to enable more precise search criteria. Let us take an example of mobile user who requests for online stock quote service. All static and dynamic parameters related to the services are embedded in the list. This is very important as the UDDI Proxy will use some of this parameter values as ranking criteria. Based on the criteria preferences defined by administrator, if the ranking feature is enabled, the UDDI Proxy will further process the list accordingly, using the embedded parameters values. Once processing is done, the new list which contains ranked and sorted services will be sent to user interface, all the parameters values will be discarded. ii. Saving Parameters Values Saving of parameters values to UDDI Server will be handled by the UDDI Proxy using the Save functions of the UDDI publishing API. For static parameters, its values can be edited only by the administrator. This can be achieved by having UDDI Proxy to display and save the parameter values directly to UDDI server. The save frequency is solely depending on the registry administrator. As for the dynamic parameters, its values will be updated each time the Proxy detect a request has been made to access the respective business or service links. If the dynamic parameter is used to store an incremental number such as vendor ranking or popularity, first the UDDI Proxy is required to read the current parameter value, increment the value by 1 before it invoke the save function. The main advantage of the first model is the criteria data are stored and bind with its associated business or service entity. This will be beneficial for private registry operator who wishes to extend UDDI capabilities to support ranking with minimal changes to his present system architecture. However, there might be certain performance issue if the Proxy accesses launch too many queries, too frequently to the UDDI server.

Model Where Parameters are retrieved from Server Logs

A private registry system normally consists of several application and server components. A typical UDDI server is often hosted together with application server and SOAP server or being part of a integrated solution package. As with the UDDI server, these servers do provide cross-language logging services for purposes of application debugging and auditing. Web service log data could provide information such as Web service usage, supporting information concerning business transaction and quality of service. These logs data could provide useful semantic information for ranking criteria. Fig.2.14 shows the components and data flow of this second model. Note this model does not support the retrieving or saving of static parameters.

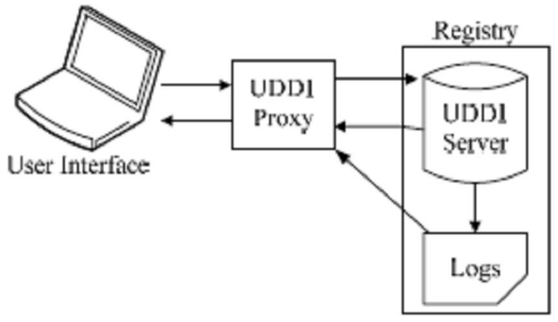


Figure 2.14 Model 2 – Parameter values to be retrieved from logs data

Retrieving Dynamic Parameter Values In this second model, we propose the creating of dynamic parameter values by extracting and processing the data from log files of SOAP server, application server and UDDI server. A function used to search, match and count for each parameter type is required within the UDDI Proxy. ii. Saving Dynamic Parameter Values Since dynamic parameters values are extracted from the log files and the log processing is handled by the respective server logging services, there will be no saving mechanism introduced here. The only important requirement is to ensure all the servers logging service are turned on, or to the minimum level where UDDI will be created within the logs. The main advantage of the second model is the criteria data can be automatically generated from the server logs. This will simplify implementation procedures and ensure data received are the most recent. Registry administrator who does not require static parameters for their criteria will find this model suitable for their need. Besides, this model can be further extended to monitor the health of registry servers as described in.

<u>Model Where Parameters are saved and Retrieved from External File</u>

In this model, researcher proposes keeping both parameter values in external files, one file for each parameter type. As shown in Fig. 2.15, the files should be accessible directly from the Proxy, outside the UDDI server. The flat ASCII file can either be in pipe-delimited or even XML format. File A is used to store values for static parameters and it can be modified by administrator only. File B is used to store values for dynamic parameters and gets updated by certain functions within the UDDI Proxy.

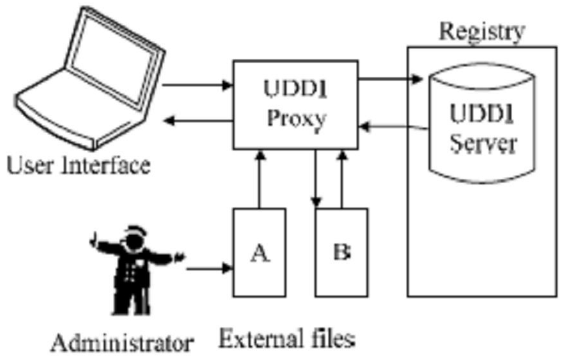


Figure 2.15 Model 3 - Parameter values to be saved and retrieved from external files

Unlike the first model where saving of parameter values will be added to existing UDDI record based on XML schema, this model will have its own data structure to store business/service parameters values. The third model introduces distributed storage of the parameters data; it has the advantages of lowering the UDDI Server load, and gives administrator more control over the external files. However, with more control available at the administrator interface, the UDDI Proxy will have to provide more complex functions to support these requirements and file handling processing. This model will best suite registry operator who has long list of criteria parameters, require full control of the parameters data, and has to generate complex criteria on the registry query results.

Claudia Diamantini, Domenico Potena, Jessica Cellini, "*UDDI registry for Knowledge Discovery in Databases services*" *2007* [18]. In this paper the authors discussed the design and implementation of the UDDI service broker, a core element of the platform. They analyze the information needed to describe a tool in our platform, showing limitations of the present UDDI standard. Then, they present the solution to overcome such limitations and to extend UDDI broker capabilities In this paper, they discuss how to extend the UDDI registry in order to manage information needed to describe a service in the KDDVM platform, focusing on the description of KDD tools. UDDI specifications define two ways to add new information into a registry. One possibility is to define a tModel in order to address, by the *overviewDoc* field, WSDL description. In this way, WSDL and UDDI work together for web services advertisement. As a matter of fact, a WSDL document defines how to invoke a service. It provides information on the data being exchanged, the sequence of messages for an operation, the location of the service and the description of bindings (e.g. SOAP or HTTP). The other way is to use a category- Bag to classify services based on their functionalities. A categoryBag is a collection of *keyedReference* structures. Each keyedReference provides a <name,value> pair, that assumes values in a particular domain described by the related tModel.

Eyhab Al-Masri and Qusay H. Mahmoud, *"Toward quality driven web Service Discovery" 2008* [3]. In this paper, the authors provide quality-driven discovery using our *Web service broker* (WSB), shown in Figure. In the WSB model, service providers publish service information in the UDDI or search engines. The WSB collects Web services disseminated throughout the Web and continuously monitors their behavior based on various QWS metrics. WSB requires no human intervention because it performs these functions automatically. Service providers can also submit their Web services to the WSB. The WSB interface lets clients articulate proper service queries based on QWS. When clients receive response messages, they can invoke services. To assess a particular Web service's quality, the service must contain at least one accessible operation - that is, it must have a valid service end point. However, a Web service might contain one or more operations but the service end point is inaccessible, so the service can't be monitored or considered serviceable. WSB performs a series of tests to determine a collected Web service's serviceability.

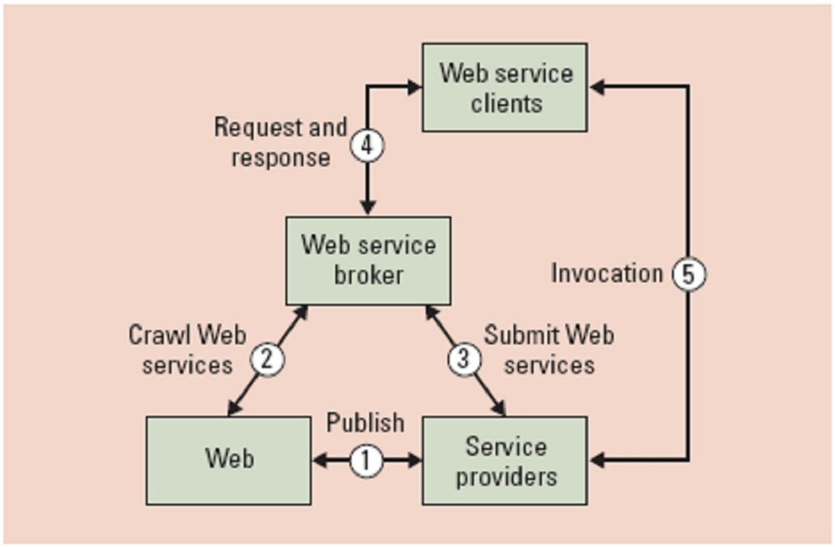


Figure 2.16 High-level architecture of WSB model.

The WSB automatically collects Web services disseminated throughout the Web and monitors their behavior using various QWS metrics. Clients use the WSB interface to enter QWS-based queries. For example, a Web service interface might contain two or more operations, but the actual service end point to invoke these operations requires authentication or contains an invalid location. This makes achieving trialability impossible. Therefore, amplifying Web services prior to any QWS monitoring can ensure their trialability and that they're serviceable.

Eyhab Al-Masri and Qusay H. Mahmoud, *"Investigating Web Services on the World Wide Web" 2008* [21]**.** In this work, the authors conduct a thorough analytical investigation on the plurality of Web service interfaces that exist on the Web today. Using their Web Service Crawler Engine (WSCE), we collect metadata service information on retrieved interfaces through accessible UBRs, service portals and search engines. This data can be used to determine Web service statistics and distribution based on object sizes, types of technologies employed, and the number of functioning services. This statistical data can be used to help determine the current status of Web services. *UDDI Business Registries (UBRs)* UBRs are used for publishing and discovering Web services into registries. There are several key UBRs that currently exist and were used for this method including: Microsoft, XMethods, SAP, National Biological Information Infrastructure (NBII), among others. Web-based crawling involves using an existing search engine API to discover WSDL files across the Web such as Google and Yahoo search APIs. Using this method, a crawler engine can continuously parse search results from an existing search engine when looking for Web services throughout their indices. This involves the use of search engine specific features to collect Web service information. For example, Google Search API provides a way to search for files with any extension such as WSDL, DISCO, or WSIL. There were several key search engines indices that were used for crawling these types of service resource including: Google, Yahoo, AlltheWeb, and Baidu. The crawling tools consist of a verifier, validator, and metadata collector. A Web service is passed to the WSCE crawler tools after a resource is examined. Crawlers are used to build the backend index for search engines by following links from one page to another. However, Web service crawling is relatively distinctive from Web page crawling

Eyhab Al-Masri and Qusay H. Mahmoud *"Discovering the Best Web Service"* 2007 [20]. This work introduces the Web Service Relevancy Function (WsRF) used for measuring the relevancy ranking of a particular Web service based on QoS metrics and client preferences. The main focus of their approach is to design an intelligent system that has the potential of examining web service's QoS properties in an open and transparent manner, and enabling clients to select the best available web service by taking advantage of client QoS preferences, Web service capabilities, and service provider features. This is achieved through the WS-QoSMan service broker. The architecture of the proposed WS-QoSMan solution is shown on

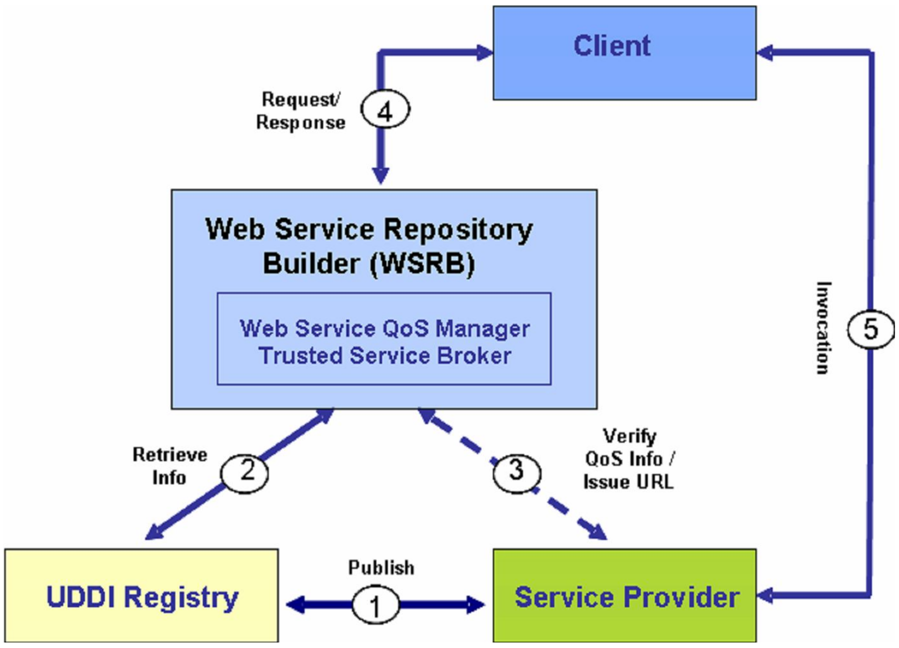

Client
Request/ Response
4
Web Service Repository Builder (WSRB)
Web Service QoS Manager Trusted Service Broker
Invocation
5
Retrieve Info
2
3
Verify QoS Info / Issue URL
UDDI Registry
Publish
1
Service Provider

Figure 2.17 Architecture based on WS-QoSMan service broker

QoSMetrics uses overviewURL to point to an XML-based file generated by WS-QoSMan and that contains QoS metrics for a specific Web service. WsRF is used to measure the relevancy ranking of a particular Web service wsi. Clients can submit their requests to WS-QoSMan (i.e. via a GUI) which will process these requests and compute WsRF values for all available services related to search query. A Web service with the highest calculated WsRF value is the most desirable and relevant to the client based on his/her preferences. In order to calculate WsRF(wsi), we need the maximum normalized value for each set of QoS parameters.

Ivan Magdalenic, Ivo Pejakovic, Zoran Skocir,Mihaela Sokic, Marina Simunic, *"Modeling ebXML Registry Service Architecture" 2003* [55]. In this paper, the authors have modeled ebXML Registry/Repository architecture and concentrates on its Registry Service part. In their implementation they have made many improvements over the existing open-source implementation. They made the ebXML Registry Services server distributed which is very important knowing that processing of the business documents is highly resource demanding. Also the system of Registry Services can share same registry items and metadata about them.

***Stefan Schulte, Melanie Siebenhaar, and Ralf Steinmetz**, " Integrating Semantic Web Services and Matchmaking into ebXML Registry" 2010* [76]. In this paper, the authors presented a solution extending the ebXML Registry by capabilities to handle and provide SWS. This includes a concept for the integration of SWS into ebXML Registry as well as a prototypical implementation using SAWSDL and the open source framework freebXML. They have proposed a quite lightweight interface for matchmakers. The interface is based on the assumption that service requests are formulated using a "query by example" approach.

Summarizing above papers, UDDI and ebXML have many things in common and can complement each other. Both technologies provide solutions to integration problems, both use XML over Internet for Message interchange, and both approaches share a common high-level architecture. Observing the e-Business world reveals the evolution from tactical systems with limited scope to strategic e-Business initiatives. This does not mean, however, that UDDI will soon be abolished and replaced by ebXML. UDDI is a well established and widely adopted standard. A multitude of experienced developers use the numerous available libraries and frameworks to guarantee short time to market for their products. In addition to those strengths, the UDDI domain is much broader than that of ebXML and its architecture is simpler and easier to handle. As a successor of other middleware technologies, UDDI excel in intra-enterprise request/response type application integration environments. The major drawbacks of ebXML are that the specification is not entirely complete and that industry support is still lacking. If industry fails to provide affordable implementations of ebXML, this standard might follow the destiny of EDIFACT, which was not widely adopted due largely to its cost. Since ebXML is powerful, implementations are likely to be complex and might not be easy to handle. Templates for the most common demands of companies might help to decrease the time-to-market for system providers that use ebXML implementations. While ebXML is always intended for e-Business, UDDI is a bottom-up technology that focuses on the technical aspects of middleware functionality. However, for many in-house projects companies do not need full grown e-Business suites. Instead, they need smaller, more reliable, and easier to handle technologies that have reached a sufficient level of maturity.

Summarizing above discusion, it can be concluded that the two emerging standards which could have very well impact on the way of conducting e-business in future are UDDI (Universal Description, Discovery and Integration) and ebXML (electronic business XML).UDDI is a vendor-sponsored initiative led by IBM, Microsoft, and Ariba, whereas, ebXML is a UN/CEFACT (United Nations center for Trade Facilitation and Electronic Business) / OASIS sponsored initiative for creating a single global electronic market. UDDI and ebXML, make it possible for business organizations to publish information on the Internet about their products and web services, where the information can be readily and globally accessed by clients who want to do business. Using UDDI based mechanism, WSCE collect metadata service information on retrieved interfaces through accessible UBRs, service portals and search engines. Broker based mechanisms allow user to specify the functional requirement and QoS parameter values for searching the services. For semantic web service discovery, DAML-S can be used. DAML-S uses semantic annotations and ontologies to relate each web service's description to a description of its operational domain. For example, a DAML-S description of a stock-reporting service might specify the data it reports, its delay versus the market, and the cost of using the service.

2.2.3 Alternative mechanisms

2.2.3.1 Federated Registry

Kaarthik Sivashanmugam, Kunal Verma, Amit Sheth, *"Discovery of Web Services in a Federated Registry Environment" 2004* [85]. They have presented the implementation of a peer-to-peer network of private, semi-private and public UDDI registries which allows transparent access to other registries based on registry federation or domains. An ontology based approach is used to classify registries and locate them based on the user requirements. They have also presented the way in which web service discovery is carried out within a federation. In their initial, naïve implementation registries could only be categorized based on business domains. Extended Registries ontology (XTRO), represented in OWL, is a comprehensive ontology containing details of Domains, Registries, Ontologies and Registry

Federation and network of relationships among them. All the classes and few important object properties in XTRO are shown in Fig. 2.18.

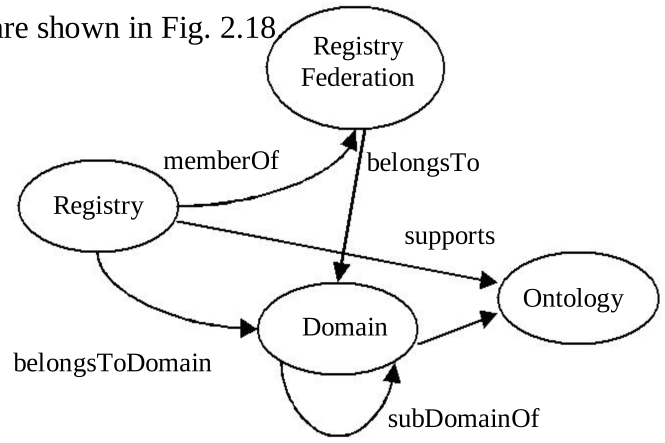


Figure 2.18 Classes and their Relationships in XTRO

Abraham Bernstein, Mark Klein, *"Discovering services: Towards High-Precision Service Retrieval" 2004* [52]. They described a novel service retrieval approach based on the sophisticated use of process ontologies. They claim that this approach offers qualitatively higher retrieval precision than existing (keyword and table based) approaches without sacrificing recall and computational scalability. In this approach, the salient behavior of a service is captured using process models, and these process models, as well as their components (subtasks, resources, etc.), are placed in the appropriate locations in the process ontology. Queries can then be defined (using a *p*rocess *q*uery *l*anguage – PQL) to find all the services whose process models include a given set of entities and relationships. The greater expressiveness of process models, as compared to keywords or tables, offers the potential for substantively increased retrieval precision, at the cost of requiring that services be modeled in this more formal way. This process-based approach offers qualitatively increased retrieval precision, and beside this it can be achieved with a reasonable expenditure of service modeling effort. The approach has the funct ıre shown in Figure.

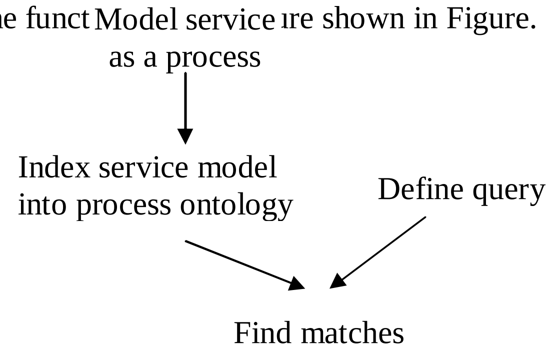

Model service as a process
Index service model into process ontology
Define query
Find matches

Figure 2.19 Service retrieval approach based on process ontology

***Jorge Cardoso and Amit Shet**, "Semantic e-Workflow Composition" 2002* [47]. They have presented a methodology and a set of algorithms for web service discovery based on three dimensions: syntax, operational metrics, and semantics. This approach allows for web service discovery not only based on functional requirements, but also on operational metrics.

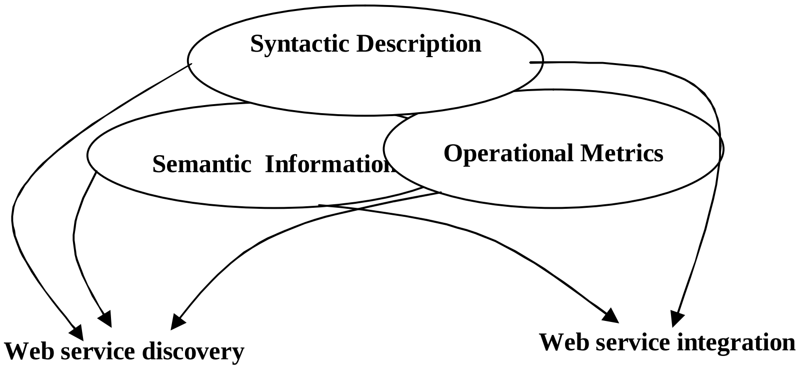


Figure 2.20 Multidimensional approach to Web Service Discovery and Integration

***Jinghai Rao, Dimitar Dimitrov, Paul Hofmann and Norman Sadehw,** "A Mixed Initiative Approach to Semantic Web Service Discovery and Composition : SAP's Guided Procedures Framework" 2006* [45]. They described a mixed initiative framework for semantic web service discovery and composition that aims at flexibly interleaving human decision making and automated functionality in environments where annotations may be incomplete and even inconsistent. Fig. 2.21 depicts overall architecture of the system.

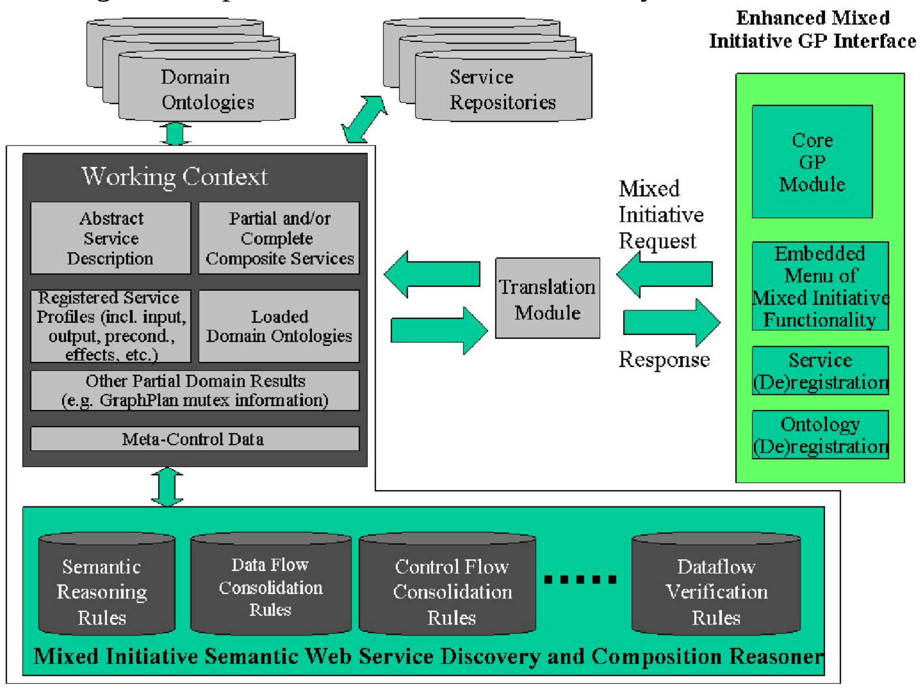

Enhanced Mixed Initiative GP Interface
Domain Ontologies
Service Repositories
Working Context
Abstract Service Description
Partial and/or Complete Composite Services
Registered Service Profiles (incl. input, output, precond., effects, etc.)
Loaded Domain Ontologies
Other Partial Domain Results (e.g. GraphPlan mutex information)
Meta-Control Data
Translation Module
Mixed Initiative Request
Response
Core GP Module
Embedded Menu of Mixed Initiative Functionality
Service (De)registration
Ontology (De)registration
Semantic Reasoning Rules
Data Flow Consolidation Rules
Control Flow Consolidation Rules
Dataflow Verification Rules
Mixed Initiative Semantic Web Service Discovery and Composition Reasoner

Figure 2.21 Architecture of *Semantic Web Service Discovery and Composition*

***Patrick C. K. Hung And Haifei Li,** "Web Services Discovery Based on the Trade-off between Quality and Cost of Service: A Token based Approach" 2003* [67]. They have proposed a token based approach for web services discovery based on the trade-off between Quality and Cost of Service (QoS and CoS) to quantify the QoS and CoS for achieving integrative solutions. In this model, the QoS relates to performance-oriented capabilities and the CoS relates to services'resource requirements. To achieve an integrative solution, both parties have to evaluate the list of QoS and CoS alternatives for obtaining an appropriate combination. One of the negotiation strategies for achieving integrative solutions for both parties is called logrolling. Logrolling is an important step in web service discovery process in which both web services providers and web services requestors can find appropriate partners.

Summarizing above papers, it can be concluded that service discovery mechanism should be based on not only functionality and QoS of the service desired by the user, but also it should allow them to specify the domain to which that service belongs. Also the user should be able to evaluate tradeoffs between QoS and CoS in selecting perfect service.

2.3 Limitations of existing mechanisms

A significant amount of literature is available on web service discovery mechanism and techniques. Still, the pros and cons of these mechanisms and techniques have not been adequately studied with respect to their performance and interface.

M. Paolucci, T. Kawamura, T. Payne, and K. Sycara [2002] focused on discovering Web services through a centralized UDDI registry. Although centralized registries can provide effective methods for the discovery of Web services, they suffer from problems associated with having centralized systems such as a single point of failure, and bottlenecks. In addition, other issues relating to the scalability of data replication, providing notifications to all subscribers when performing any system upgrades, and handling versioning of services from the same provider have driven researchers to find other alternatives.

Jorge Cardoso and Amit Sheth [2002] presented a methodology and a set of algorithms for Web service discovery based on three dimensions: syntax, operational metrics, and semantics.

This approach allows for Web service discovery not only based on functional requirements, but also on operational metrics.

Mario Schlosser, Michael Sintek, Stefan Decker, Wolfgang Nejdl [2002] proposed a graph topology which allows for very efficient broadcast and search, and provide an efficient topology construction and maintenance algorithm which, crucial to symmetric peer-to-peer networks, does neither require a central server nor super nodes in the network.

Bernstein, Abraham, and Mark Klein [2002] described a novel service retrieval approached based on the sophisticated use of process ontologies. This approach offers qualitatively higher retrieval precision than existing (keyword and table based) approaches without sacrificing recall and computational tractability/scalability.

Patrick C. K. Hung And Haifei Li [2003] proposed a token based approach for web services discovery based on the trade-off between Quality and Cost of Service (QoS and CoS) to quantify the QoS and CoS for achieving integrative solutions. One of the negotiation strategies for achieving integrative solutions for both parties is called logrolling. Logrolling is an important step in web service discovery process in which both web services providers and web services requestors can find appropriate partners.

D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGunneess, B. Barsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara [2004] and D. Roman, H. Lausen, and U.Keller [2004] attempted to provide a formal way of expressing service provider's capabilities and user's requirements. These initiatives are mainly focused on knowledge representation aspects. Apart from knowledge representation, the web service discovery is a complex task and need to consider the context of its availability and usability.

U. Keller, R. Lara, A. Polelres, I. Toma, M. Kifer, and D. Fensel [2004] described different levels of service matches. It is understood that service matches are mandatory but not sufficient for Web service discovery.

K. Sivashanmugam, K. Verma, and A. Sheth [2004] proposed METEOR-S Web Service Discovery Infrastructure(MWSDI), an ontology based infrastructure to provide access to private and public registries divided based on business domains and grouped into federations for enhancing the discovery process. METEOR-S provides a discovery mechanism for publishing Web services over a federated registry sources but, similar to the centralized registry environment, it does not provide any means for advanced search techniques which are essential for locating appropriate business applications. In addition, having a federated registry environment can potentially provide inconsistent policies to be employed which will significantly have an impact on the practicability of conducting inquiries across the federated environment and can at the same time significantly affect the productiveness of discovering Web services in a real-time manner across multiple registries.

K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller [2004] presented METEOR-S Web Services Discovery Infrastructure (MWSDI), a scalable infrastructure for semantic publication and discovery of Web services. We have presented two algorithms for semantic publication and discovery using WSDL descriptions.

Fatih Emekci, Ozgur D. Sahin, Divyakant Agrawal, Amr El Abbadi [2004] proposed a structured peer-to-peer framework for web service discovery in which web services are located based on both service functionality and process behavior. In addition, they integrate a scalable reputation model in this distributed peer-to-peer framework to rank web services based on both trust and service quality.

Shou-jian Yu, Xiao-kun Ge, Jing-zhou Zhang, Guo-wen Wu [2006] presented a flexible Web service discovery architecture by combining semantic Web service with P2P networks. This system does not need a central registry for Web service discovery. They use an ontology-based approach to capture real world knowledge for semantic service annotation.

Eyhab Al-Masri and Qusay H. Mahmoud [2007] proposed a solution by introducing the Web Service Relevancy Function (WsRF) used for measuring the relevancy ranking of a particular Web service based on QoS metrics and client preferences for the purpose of finding the best available Web service during Web services' discovery process based on a set of given client QoS preferences or QoS search criteria.

Gang Zhou, Jianjun Yu, Rui Chen, Hui Zhang [2007] proposed a peer-to-peer framework, which adopts an enhanced Skip Graph named ServiceIndex as the overlay network for service discovery. To guarantee discovery efficiency, ServiceIndex schemed WSDL-S (Web Services Semantics) as Semantic Web Services description language and extracted its semantic attributes as indexing keys in Skip Graph.

Jacek Kopeck´y [2007] intended to research an approach to SWS offer discovery that will significantly simplify the needed semantic descriptions and thus help ease the adoption of SWS technologies in the industry.

Qiang He, Jun Yan, Yun Yang, Ryszard Kowalczyk, Hai Jin [2008] proposed a peer-to-peer based decentralized service discovery approach named Chord4S. To improve data availability, Chord4S distributes the descriptions of functionally-equivalent services. An efficient routing algorithm is provided to facilitate queries of multiple candidate service providers.

Eyhab Al-Masri and Qusay H. Mahmoud [2008] proposed Web Service Crawler Engine (WSCE), a crawler that is capable of capturing service information from various accessible resources over the Web, to help in conducting investigation of Web services on the Web.

Shuiguang Deng, Zhaohui Wu, Jian Wu and Ying Li [2008] proposed a two-phase semantic-based service discovery mechanism to discover services in an accurate, efficient and automatic way. Compared to other approaches, the new method has two salient characteristics: (a) it takes into account the interface dependencies implied within an operation while performing matchmaking; (b) it supports two-level matchmaking, namely operation matchmaking and operation-composition matchmaking.

2.4 The Pilot study

Prior to the main research work, a pilot survey study was conducted in which a questionnaire was filled up by the around 220 service consumers from different groups of people like students, teachers, homemakers, software engineers etc. Sampling technique used for conducting the pilot study was Convenience Sampling and Purposive Sampling. Out of 220 consumers, 20 were service engineers who need to discover the service over the web for integrating it in their applications for some system related tasks whereas 197 were the direct users of the service who utilizes online services either for shopping, booking, bank transactions, bill payments etc. for their own purpose. Among 200 people who filled up the pilot study survey questionnaire, 3 people had never used any web service. Out of 200 customers, 89 were satisfied, 81 were not satisfied and 30 can’t say anything about the online services use available over the internet.

It is very obvious that a user's perception varies from person to person. However, based on the experiences as an end user who uses online services available over the internet, following result is inferred on how they perceive the quality of service as compared to their expectation.

Service Domain and No. of Customers	No. of Customers Satisfied with Response Time	Overall all Satisfied Customers
Shopping (68)	16	31
Booking (57)	7	10
Bank Transactions (44)	11	20
Bill Payments (28)	14	28

Table 2.1 : Relation between Good Response Time and Satisfied Customer

From Table 2.1, the computed correlation coefficient (0.99) is positive and significant. Hence there is a strong relationship between Good Response Time and Satisfied Customer.

Service Domain and No. of Customers	No. of Customers Satisfied with Reliability	Overall all Satisfied Customer
Shopping (68)	66	31
Booking (57)	28	10
Bank Transactions (44)	40	20
Bill Payments (28)	26	28

Table 2.2 : Relation between High Reliability and Satisfied Customer

From Table 2.2, the computed correlation coefficient (0.56) is positive. Hence there is a good relationship between High Reliability and Satisfied Customer.

Service Domain and No. of Customers	No. of Customers Satisfied with Availability	Overall all Satisfied Customer
Shopping (68)	60	31
Booking (57)	18	10
Bank Transactions (44)	30	20
Bill Payments (28)	27	28

Table 2.3 : Relation between High Availability and Satisfied Customer

From Table 2.3, the computed correlation coefficient (0.76) is positive and significant. Hence there is a strong relationship between High Availability and Satisfied Customer.

Service Domain and No. of Customers	No. of Customers Satisfied with Price	Overall all Satisfied Customer
Shopping (68)	45	31
Booking (57)	21	10

Bank Transactions (44)	39	20
Bill Payments (28)	27	28

Table 2.4 : Relation between Good Price and Happy Customer

From Table 2.4, the computed correlation coefficient (0.66) is positive and significant. Hence there is a good relationship between Good Price and Satisfied Customer.

Service Domain and No. of Customers	**No. of Customers Satisfied with Response Time**	**No. of Customers Satisfied with Reliability**	**No. of Customers Satisfied with Availability**	**No. of Customers Satisfied with Price**	**Overall Satisfied Customer**
Shopping (68)	16	66	60	45	31
Booking (57)	7	28	18	21	10
Bank Transactions (44)	11	40	30	39	20
Bill Payments (28)	14	26	27	27	28

Table 2.5 : Customers in different service domain Satisfied with different QoS

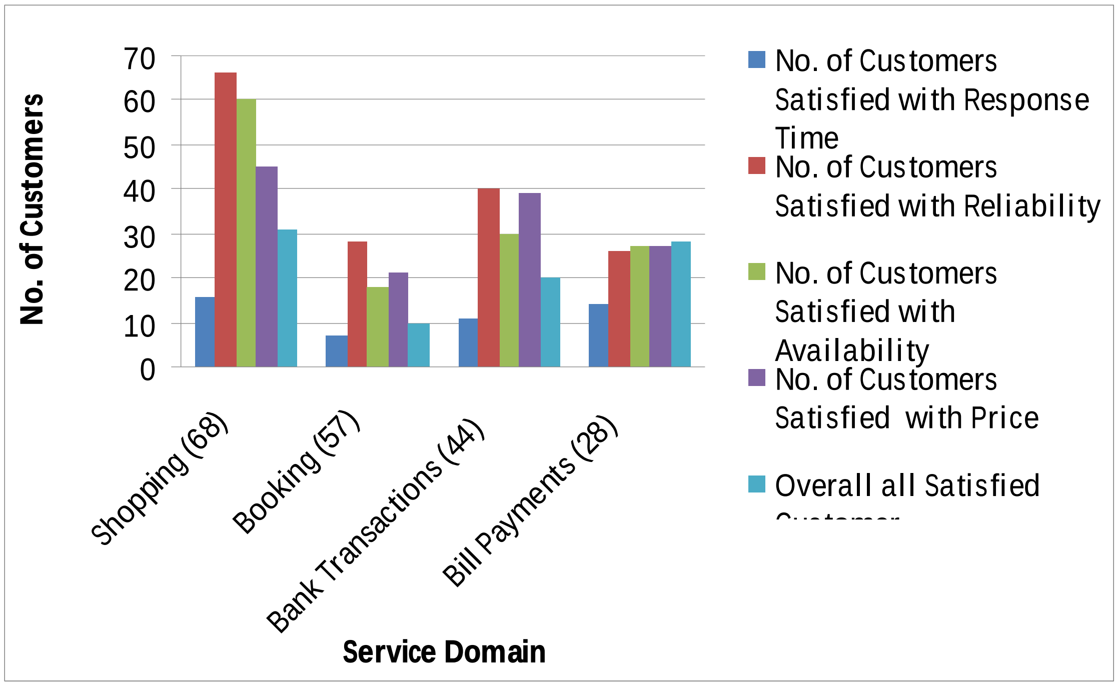


Figure 2.22 : Customers in different service domain satisfied with different QoS

In Figure 2.22, a graph of number of customers belonging to different service domain satisfied with different QoS parameter values shows that, customers are not satisfied only with the service functionality but they also want a desired level of QoS parameter value. E.g. for the customers performing bank transactions online, reliability may be the highest priority QoS parameter than any other parameter and so on.

Based on consumer's experiences as a service engineer who needs to find appropriate web services available over the internet, while designing and developing software applications, following result is inferred on how the quality of service affects the business application in terms of complaints received from the customers.

Complaint type	**Often**	**Sometimes**	**Rarely**	**Never**
Slow response	13	6	1	0
Service temporarily unavailable	10	6	4	0
Transaction not completed successfully	11	5	4	0
Costing Issue	10	5	5	0

Table 2.6 : Service Engineers receiving complaints about service

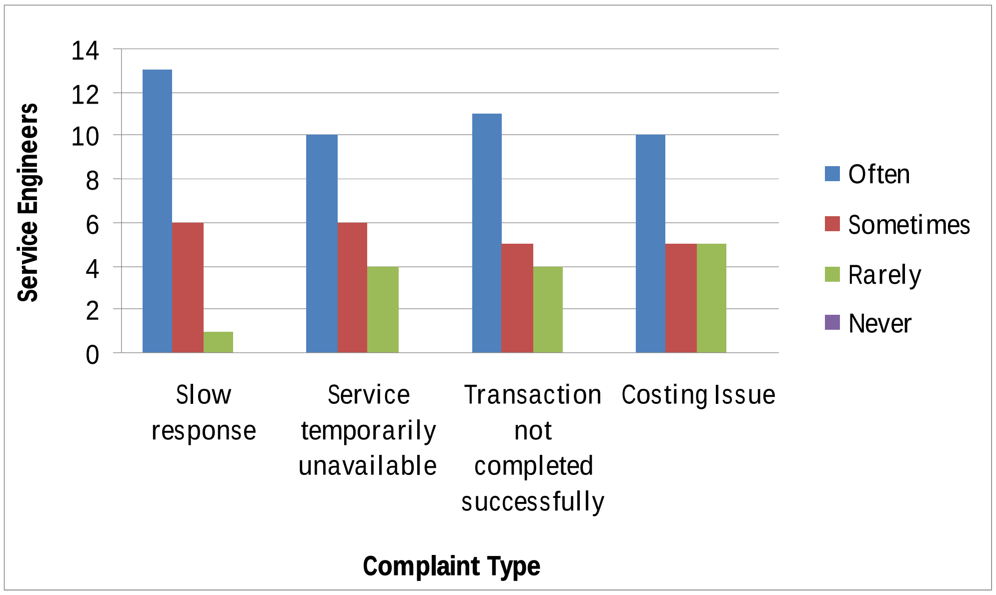


Figure 2.23 : Service Engineers receiving complaints about service

In Figure 2.23, it is observed that more number of service engineers are facing the complaints from the customers for ‘Slow response’, ‘Service temporarily unavailable’ and ‘Transaction not completed successfully’. Many service engineers also fill that the cost of integrating a web service in an application should not be more than developing the whole application on their side. Out of 20 engineers, not a single engineer is having zero complaints about service. QoS parameters have become equally important for the customers along with the service functionality.

The pilot study survey result obtained as above signifies the study of existing web service discovery mechanisms and proposing an efficient mechanism which should be able to discover the most appropriate web service as per the consumer's requirement of functionality as well as quality of service (QoS) and the priority of QoS.

2.5 The present study

The present research study has the objective to identify and evaluate the significant web service discovery technique which will be efficient in discovering the most appropriate web service according to the consumer's requirement of functionality as well as quality.

In the present research study, the approach is to return maximum number of relevant web services of desired functionality and quality efficiently and rank them according to users' preference of selecting his choice of QoS .

Thus, the study suggests to evaluate other implementations of algorithms for matching, ranking and selecting web services efficiently. The evaluation of performance is done on various parameters identified under the environmental limitations.

A new tool having a great interface for specifying service requirement, choosing right QoS values, and setting preference of QoS for ranking the services is also proposed. Algorithms are proposed for matching, ranking and selecting the web services.

Chapter 3

Service Registries

Service registries are essential part of a Service Oriented Architecture because of three reasons. First, a registry serves as a system of record for the enterprise's web services and becomes the central reference for the distributed and difficult-to-find services. Second, a registry is a place where service provider can publicize services and consumers can discover them. And third, it also controls and governs the availability of services, managing versioning and ensuring compliance with enterprise and external requirements.

Java API for XML Registries (JAXR)

JAXR is a new API that is under development under the Java Community Process (JCP) and the first public draft of the specification was released on August 10, 2001. Currently, there are several business registries available in the market. Few of them are UDDI, ebXML, ISO 11179, OASIS and eCo Framework. For accessing these registries, APIs vary considerably and this makes difficult for writing portable client programs. The JAXR specifications tries to unify access to these registries and probably the future registries by defining a new Java API.

JAXR Architecture

The high level architecture of JAXR consists of the following parts :

- *A JAXR client* : This is a client program that uses the JAXR API to access a business registry via a JAXR provider.

- *A JAXR provider* : This is an implementation of the JAXR API that provides access to a specific registry provider or to a class of registry providers that are based on a common specification.

A JAXR provider implements two main packages :

- *javax.xml.registry*, which consists of the API interfaces and classes that define the registry access interface.
- *javax.xml.registry.infomodel*, which consists of interfaces that define the information model for JAXR. These interfaces define the type of objects that reside in a registry and how they relate to each other. The basic interface in this package is the *RegistryObject* interface. Its subinterfaces include *Organization*, *Service* and *ServiceBinding*.

The most basic interfaces in the *javax.xml.registry* package are

- Connection. The Connection interface represents a client session with a registry provider. The client must create a connection with the JAXR provide in order to use a registry.
- *RegistryService*. The client obtains a *RegistryService* object from its connection. The *RegistryService* object in turn enables the client to obtain the interfaces it uses to access the registry.

The primary interfaces, also part of the *javax.xml.registry* package are

- *BusinessQueryManager*, which allows the client to search a registry for information in accordance with the *javax.xml.registry.infomodel* interfaces.
- *BusinessLifeCycleManager*, which allows the client to modify the information in a registry by either saving it or deleting it.

When an error occurs, JAXR API methods throw a *JAXRException* or one of its subclasses.

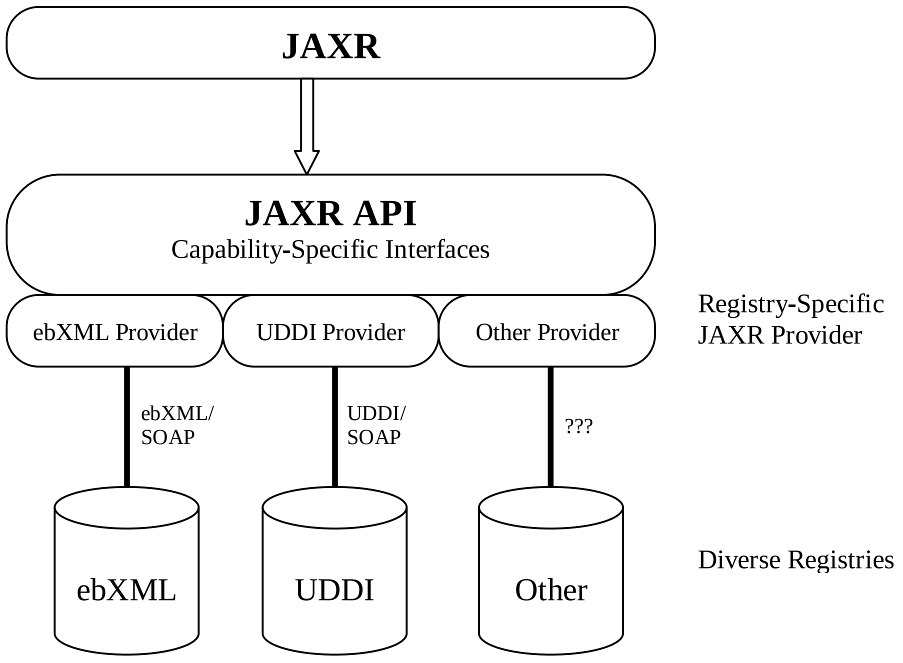


Figure 3.1 JAXR Architecture

Figure 3.1 illustrates the architecture of JAXR a JAXR client uses the capability level0 interfaces of the JAXR API to access the JAXR provider. The JAXR provider in turn accesses a registry. The Application Server supplies a JAXR provider for UDDI registries.

We present here the registry approach for web service discovery based on the following two available service registries.

- Universal Description, Discovery and Integration (UDDI) Registry
- Electronic Business XML (ebXML) Registry

3.1 Universal Description, Discovery and Integration (UDDI) Registry

The Universal Description, Discovery and Integration (UDDI) is the specification of a multi-purpose, platform independent, web-service definition registry. UDDI is an OASIS standard that allows users to enquire about services available on a given network and also let the developers publish their services by specifying in the registry the information related to these services (like their operations, prerequisites or specification). The purpose of UDDI compliant registries is to provide a service discovery platform on the World Wide Web. Service discovery is related to being able to advertise and locate information about different technical interfaces exposed by different parties. Services are interesting when you can discover them, determine their purpose, and then have software that is equipped for using a particular type of web service and derive benefit from a service. A UDDI compliant registry provides an information framework for describing services exposed by any entity or business. In order to promote cross platform service description, this description is rendered in cross-platform XML.

The registry itself is based on multiple web protocol standards and technologies like HTTP, XML, and SOAP. UDDI defines a web service discovery protocol, which let the clients find web services and a web service description format, which lets clients understand what those web services do. A UDDI registry typically contains metadata for a service embodied within a Web Service Description Language (WSDL) document. The UDDI data structures provide a framework for the description of basic business and service information, and architect an extensible mechanism to provide detailed service access information using any standard description language. Using the information provided in a UDDI registry, three types of searches can be performed as :

1. A white pages search returns basic information such as address, contact, and identifiers about a company and its services.
2. A yellow pages topical search retrieves information according to industrial categorizations and taxonomies, such as the NAICS, ISO3166, and UNSPSC classification systems.
3. A green pages service search retrieves technical information about web services, as well as information describing how to execute these services.

3.1.1 Data Model of UDDI

Understanding how providers and services are represented in a web service environment is an essential part of using UDDI services. Although the UDDI API specification provides a framework within which to perform this modeling task, each UDDI Services deployment requires organization-specific definition of four core entities. The information about web services in a UDDI registry includes a description of the business and organizations that provide the services, a description of a service's business function, and a description of the technical interfaces to access and manage those services. A UDDI registry consists of instances of four core data structures including the *businessEntity*, the *businessService*, the *bindingTemplate* and the *tModel*. The four core structures and their relationships are shown in following Figure 3.2. This information comprises everything a user needs to know to use a particular web service.

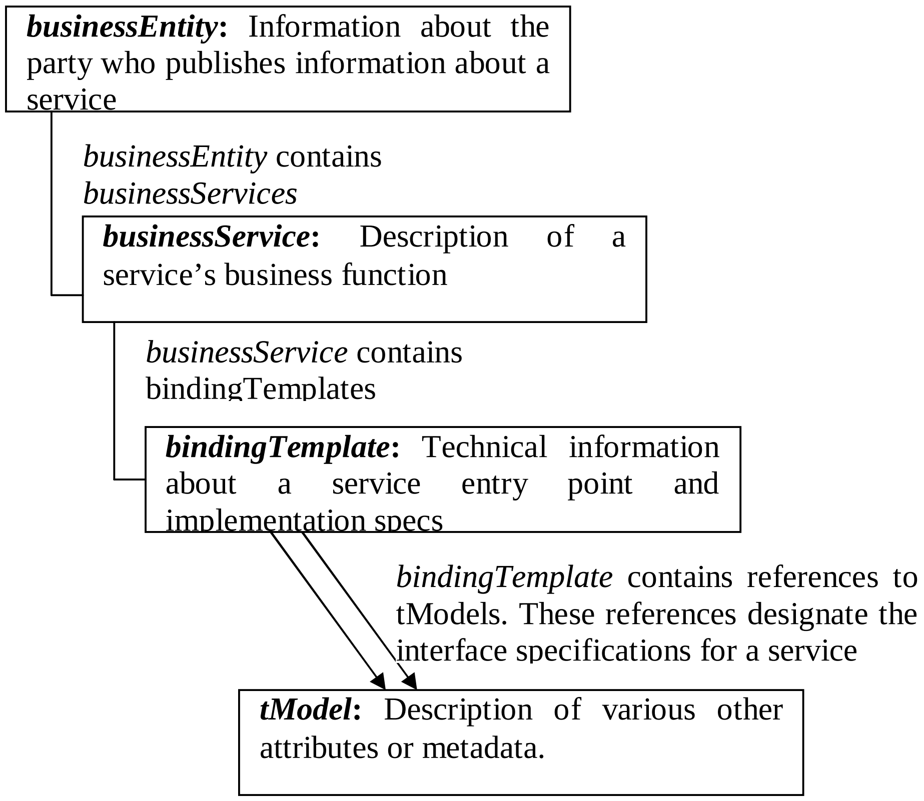

***businessEntity*:** Information about the party who publishes information about a service
businessEntity contains *businessServices*
***businessService*:** Description of a service's business function
businessService contains bindingTemplates
***bindingTemplate*:** Technical information about a service entry point and implementation specs
bindingTemplate contains references to tModels. These references designate the interface specifications for a service
***tModel*:** Description of various other attributes or metadata.

Figure 3.2 UDDI core data structures

1. ***businessEntity* -** Each *businessEntity* entity contains descriptive information about a business or organization and, through its contained *businessService* entities, information about the services that it offers. From an XML standpoint, the *businessEntity* is the top-level data structure. Each contained *businessService* describes a logical service offered by the business or organization. Similarly, each *bindingTemplate* contained within a given *businessService* provides the technical

description of a web service that belongs to the logical service that is described by the *businessService*.

2. ***businessService* –** Each *businessService* contains descriptive information about a group of related technical services including the groupname, description and category information. A *businessService* acts as a container for one or more *bindingTemplate*s.
3. ***bindingTemplate* –** Each *bindingTemplate* contains information needed to invoke or bind to a specific service. This information includes the service URL, routing and load balancing facilities and references to interface specifications contained in a corresponding *tModel*.
4. ***tModel* -** *tModel*s represent unique concepts or constructs. They are used to describe compliance with a specification, a concept or a shared design. *tModel*s are used to represent technical specifications such as service types, bindings and protocols. Also *tModel*s are used to implement category systems that are used to categorize technical specifications and services. When a particular specification is registered in the UDDI registry as a *tModel*, it is assigned a unique key, called a *tModel*Key. This key is used by other UDDI entities to reference the *tModel*, for example to indicate compliance with the specification. Each specification *tModel* contains an *overviewURL*, which provides the address of the specification itself e.g. a WSDL document.
5. ***tModels*' CategoryBags –** Additional metadata can be associated with a specification *tModel* using any number of identifier and category systems. Identifiers are grouped in a construct called an *identifierBag*, and categories are grouped in a construct called a *categoryBag*. These bags contain a set of *keyedReference* elements. Each keyedReference specifies the *tModel*Key of the category system *tModel* and a name/value pair that specifies the metadata. The metadata values specified in *keyedReference* elements can be used as selection criteria when searching UDDI.

3.1.2 Design Principles in UDDI

Each core data structure of UDDI is used to represent specific type of data, arranged in the relationship as shown in Figure 3.2 – UDDI core data structures. A particular instance of an individual fact or set of facts is expressed using XML according to the definition of these core types. For instance, two separate businesses may publish information in a UDDI registry about web services they offer. Information describing each business and its web services all exist as separate instances of the core data structures stored within the UDDI registry. Instances of many data structures in UDDI are kept separately and are accessed individually by way of unique identifiers called keys. An instance in the registry gets its keys at the time it is first published. Publisher may assign the keys; if they don't, the UDDI node must assign them.

3.1.3 UDDI services and API sets

This specification presents APIs that standardize behavior and communication with and between implementations of UDDI for the purposes of manipulating UDDI data stored within those implementations. The UDDI API is divided into two main components: the inquiry API and the publisher API. Clients access information contained in the UDDI registry using the inquiry API. Publishers of web services use the publisher API to enter and modify publisher information in the UDDI registry. Both the inquiry API and the publisher API take the form of an XML message that is placed within the body of a SOAP message envelope. The receiver URL of the SOAP message is the UDDI site. Once the UDDI site receives an enquiry SOAP message from a client, the UDDI site retrieves the requested information from the UDDI registry, which is returned to the client in the form of a SOAP message. The client receives the SOAP message and retrieves the response from the body of the SOAP message.

There are three patterns in which a client can query a UDDI registry: the browse pattern, the drill-down pattern and the invocation pattern.

1. **The browse pattern** - The browse pattern typically involves starting with some broad information, performing a search, finding general result sets and then selecting more specific information for drill-down. Browse pattern inquiries use the find_xx API

calls, where xx is a type of information contained in a UDDI registry. For example, a client may want to search for businesses whose names begin with a sequence of characters.

2. **The drill-down pattern** - The drill-down pattern is used once a client narrows choices to a selected group of candidates. Let's say that the browser pattern search returned all businesses whose names begin with ABC. The drill-down pattern uses the get_xx API calls where xx represents a specific kind of information about a particular business.
3. **The invocation pattern –** The invocation pattern is the third party inquiry pattern. The invocation pattern is used to prepare the client application to use the web services found by inquiring the UDDI site. This process is called binding and requires the client application to bind data obtained from the UDDI registry for a particular web service.

♦ **UDDI Invocation Model and UDDI Inquiry API**

To invoke a specific web service using information from a UDDI registry, a caller typically follows these steps:

1. Locates the *businessEntity* information registered for the business exposing the web service.
2. Discovers additional details about the web service by accessing the *businessService* structure contained within the *businessEntity* structure. From there, the caller selects the appropriate *bindingTemplate* to use.
3. Uses the technical information contained in the *tModel* corresponding to the selected *bindingTemplate* to build the client that will access the web service.

The UDDI inquiry API consists of operations that enable you to browse a registry and to traverse a registry in order to obtain information about specific businesses and services**.** Table 3.1 shows the inquiry API calls that a UDDI registry must support.

Table 3.1 UDDI Inquiry API Methods

Method	**Description**
find_binding	Used to locate binding within or across one or more registered *businessService*s.
find_business	Used to locate information about one or more businesses.
find_relatedBusinesses	Used to locate information about *businessEntity* registrations that are related to specific business entity whose key is passed in an enquiry.
find_service	Used to locate specific services within the registered business entities.
find_*tModel*	Used to locate one or more *tModel* information structure.
get_bindingDetail	Used to get *bindingTemplate* information suitable for making service requests.
get_businessDetail	Used to get the *businessEntity* information for one or more businesses or organizations.
get_businessDetailExt	Used to get extended *businessEntity* information.
get_serviceDetail	Used to get full details for a given set of registered *businessService* data.
get_*tModel*Detail	Used to get full details for a given set of registered *tModel* data.

♦ **UDDI Publication - Authentication Model and UDDI Publisher API**

A service provider makes services available to clients by publishing services on UDDI site using publishing API calls. Typically, a service provider selects a UDDI operator site to publish its services and update registered services. The key operating principal for the UDDI Publishers’ API is to allow only authorized individuals to publish or change information within the UDDI business registry. Each individual implementation of the distributed UDDI business registry maintains a unique list of authorized parties and tracks which individuals create each *businessEntity* or *tModel*. Further changes and deletions are allowed only if a change request, through API call, is made by the same individual who created the information.

The UDDI Publisher API consists of operations for creating, reading, updating, and deleting the information stored in UDDI registry. A caller can use these operations to register and/or modify any number of businesses or services. Table 3.2 shows the publisher API calls that a UDDI registry must support.

Table 3.2 UDDI Publisher API Methods

Method	**Description**
add_publisherAssertions	Causes one or more *publisherAssertion*s (the relationship that one *businessEntity* has with another *businessEntity*) to be added to an individual publisher's collection of assertions
delete_binding	Causes one or more instances of *bindingTemplate* data to be deleted from the registry
delete_business	Used to delete one or more business registrations from a UDDI registry
delete_publisherAssertions	Causes one or more *publisherAssertions* to be deleted from a publisher's collection of assertions
delete_service	Used to delete one or more *businessService* elements from a UDDI registry
delete_*tModel*	Used to logically delete one or more *tModel* structures
discard_authToken	Used to inform a node that the passed authorized token is to be discarded
get_assertionStatusReport	Reports the status of current and outstanding publisher assertions that involve any of the business registration managed by a publisher

Method	**Description**
get_authToken	Used to obtain an authentication token
get_publisherAssertions	Used to obtain the full set of publisher assertions associated with a publisher
get_registeredInfo	Used to obtain an abbreviated list of all *businessEntity* and *tModel* data for a publisher
save_binding	Used to save or update a complete *bindingTemplate* element
save_business	Used to save or update information about a complete *businessEntity* structure
save_service	Adds or updates one or more *businessService* elements
save_*tModel*	Adds or updates one or more registered *tModel* elements
set_publisherAssertions	Used to replace all of the assertions associated with a publisher

3.2 Electronic Business XML (ebXML) Registry

Electronic business XML (ebXML) is a set of specifications that allow businesses to collaborate. ebXML enables a global electronic marketplace where business can meet and transact with the help of XML-based messages. The businesses may be geographically located anywhere in the world and could be of any size to participate in the global marketplace. The ebXML was created in 1999 as a joint partnership by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS). The current membership includes representation from more than 2000 businesses, governments, institutions, standard bodies and individuals.

The ebXML framework defines specifications for the sharing of web-based business services. It includes specifications for a message service, collaborative partner agreements, core components, business process methodology, a registry and a repository.

ebXML defines a registry and a repository where businesses can register themselves by providing their contact information, address and so on through a standard document format. Such information is called the core component. Once the business submits core components, it can further supply the information about its products and services. After a business has registered with the ebXML registry, other partners can look up the registry to locate that business. Once a business partner is located, the core components of the located business are downloaded. The buyer may then download the technical specifications for the service. Once the buyer is satisfied with the fact that the seller service can meet its requirements, it negotiates a contract with the seller. Such collaborative partner agreements are defined in ebXML. Once both the parties agree on contract terms, they sign the agreements and do a collaborative business transaction by exchanging their private documents. The ebXML provides a marketplace and defines several XML-based documents for business to join and transact in such a marketplace. the ebXML Registry vision is to provide generic, extensible, secure, federated information management.

The ebXML Registry provides a set of services that enable sharing of information between interested parties for the purpose of enabling business process integration between such parties based on the ebXML specifications. The shared information is maintained as objects in a repository and managed by the ebXML Registry Services.

3.2.1 ebXML Registry Data Model

The ebXML Registry Information Model (or RIM) defines what metadata and content can be stored in the registry. An ebXML Registry is capable of storing any type of electronic content such as XML documents, text documents, images, sound and video. Instances of such content are referred to as a *RepositorytItems*. *RepositorytItems* are stored in a content repository provided by the ebXML Registry. In addition to the *RepositoryItems*, an ebXML Registry is also capable of storing standardized metadata that MAY be used to further describe *RepositoryItems*. Instances of such metadata are referred to as a *RegistryObjects* (or one of its sub-types). *RegistryObjects* are stored in the registry provided by the ebXML Registry. ebXML Registry stores any type of content as

RepositoryItems in a repository and stores standardized metadata describing the content as *RegistryObjects* in a registry.

3.2.2 ebXML Registry Architecture

The ebXML Registry architecture consists of an ebXML Registry and ebXML Registry Clients. The Registry Client interfaces may be local to the registry or local to the user. Registry architecture supports three possible topologies with respect to the Registry and Registry Clients. In the first topology, Registry provides a web based “thin client” application for accessing the Registry that is available to the user using a common web browser. In this scenario the Registry Client interfaces reside across the internet and are local to the Registry from the user’s view. In the second topology, the user is using a “fat client” Registry Browser application to access the registry. In this scenario the Registry Client interfaces reside within the Registry Browser tool and are local to the Registry from the user’s view. The Registry Client interfaces communicate with the Registry over the internet in this scenario. A third topology made possible by the registry architecture is where the Registry Client interfaces reside in a server side business component such as a Purchasing business component. In this topology there may be no direct user interface or user intervention involved. Instead the Purchasing business component may access the Registry in an automated manner to select possible sellers or service providers based current business needs. Clients communicate with the Registry using the ebXML Messaging Service in the same manner as any two ebXML applications communicating with each other.

3.3 UDDI Vs ebXML Registry – A Comparative Study

Table 3.2 UDDI Vs ebXML Registry

ebXML Registry and Repository	**UDDI Registry**
Has an integrated registry and repository. Can store content as well as metadata	Has no repository. Cannot store content. Can only store metadata about (or pointers to) content.

Design center is to provide secure, federated information management of any type of artifact	Design center is to be like yellow/white pages for listing businesses and services
Protocols and information model is generic and extensible	Protocols and information model is focused and specific
Supports multi-registry topologies using loosely coupled federation with optional selective replication.	Supports multi-registry topologies using replication of every transaction to all participating registries.

ebXML has always been designed for the management of large amounts of complex information using standardized and extensible metadata. Also it has extensible data-model. But, ebXML is a younger technology than UDDI. It’s also a more complex specification that covers a lot more features than the ones we currently need for our service registry. In this study, currently only the “registry” part of the specification is basically required and not the “repository” one. Also, UDDI seems to have a larger user base than ebXML. From research view point, UDDI is probably a more accessible technology to facilitate the implementation of the Web Service Discovery tool that is designed during the research study.

Chapter 4

UDDI based Web Service Discovery Mechanism

4.1 Why UDDI based mechanism ?

Universal Description Discovery & Integration (UDDI) registry provides a centralized approach in service-oriented architecture and the focus of Universal Description Discovery & Integration (UDDI) is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other web services providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access those services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a web services-based software environment for both publicly available services and services only exposed internally within an organization. [2]. In the same centralized approach, if we compare UDDI with other registry ebXML, it is observed that ebXML as always been designed for the management of large amounts of complex information using standardized and extensible metadata; Also it has extensible data-model and is more like repository rather than only registry, whereas in UDDI, protocols and information model is focused and specific and is only registry not repository. One more research influencing factor is that ebXML is a younger technology than UDDI and UDDI seems to have a larger user base than ebXML.

UDDI is a vendor-sponsored registry standard which has emerged taken a dominant role in standardization process of registries. UDDI was the brain child of Ariba, IBM, Intel, Microsoft and SAP. In 2002, the OASIS standards group took over UDDI from UDDI.org. BEA, Cincom, CA, E2Open, Enthrust, Fujitsu, HP, IBM, Intel, IONA, Microsoft, Novell, Oracle, SAP, Sun Microsystems and hundreds of other companies have endorsed it. Moreover, no fees or licenses are required to use this technology. It has the following benefits as:

- It is a standardized, transparent mechanism for describing services.
- It describes simple methods for invoking the service.
- It specifies an accessible central registry of services.

The reason that UDDI is acceptable to all the vendors is that it is built on the same SOAP standards that ordinary web services use. This means that a registry can be written in and accessed by any computer language running on any hardware platform running any operating system. Every vendor is able to create tools to interact with these registries.

4.2 The Approach

UDDI contains a number of specifications that describe how a registry stores data and how it can be accessed. Four main specifications of UDDI are as follows :

- The data structure specification describes what kind of data is stored in UDDI. The UDDI data structure is based on XML and described through an XML Schema. This schema is actually published as a separate document available from the UDDI web site.
- The programmer's API specification contains how an UDDI registry can be accessed. There are two types of API, **publishing functions** and **inquiry functions**. The publishing functions are used to create and update existing entries in the registry. The inquiry functions are all read-only and allow the existing entries to be queried programmatically. The API is programming language-independent. This is accomplished by describing the request and response data in terms of an XML document. These request and response structure map the actual content of the registry quite closely. The existing registries offer access via SOAP over HTTP which means that request and response XML data is wrapped into SOAP envelopes. The enquiry functions are available over HTTP, whereas the publishing functions are accessible via HTTPS and require a user ID and

password to be sent along with each request. Each UDDI registry provided ways for a user to obtain a valid user ID and password.

- The replication specification contains descriptions of how registries replicate information among themselves. This information is only needed for those who want to implement their own registry and integrate it with other existing registries.
- Finally, there is the Operator's specification, which is only for those who are implementing or running a UDDI registry. It defines policies for security and for data management. This specification does not make it compulsory for an operator to follow a certain policy, instead it requires that each operator publish what policies are enabled and enforced.

The following table shows the type of elements that exists in the registry together with some of the API functions that are defined for them.

Table 4.1 UDDI elements and API functions for them

Element Type	**Find Method**	**Get Method**	**Save Method**	**Delete Method**
<businessEntity>	find_business()	get_businessDetail()	save_business()	delete_business()
<businessService>	find_service()	get_serviceDetail()	save_service()	delete_service()
<bindingTemplate>	find_binding()	get_bindingDetail()	save_binding()	delete_binding()
<tModel>	find_ tModel()	get_tmodelDetail()	save_tModel()	delete_tModel()

Existing web services architecture comprises three roles: Web service Provider, Web service Consumer and Universal Description, Discovery and Integration (UDDI) registry as shown in Figure 4.2.

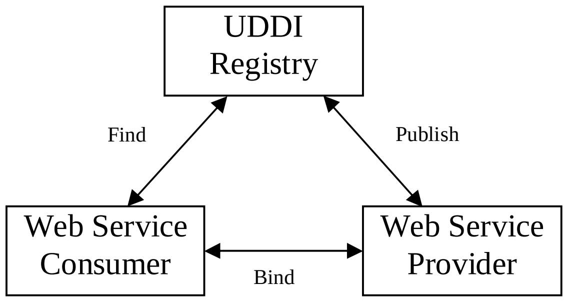


Figure 4.1: Existing Web service Architecture

The Web service Provider publishes a description of the service in the UDDI registry, as well as details of how to use the service. UDDI registries use an XML-based language, Web Services Description Language (WSDL), to describe a Web service, the location of the service and methods the service exposes. The Web service Consumer uses the UDDI to find an appropriate service that meets its requirements using the information provided with the services, chooses one service manually, and invokes the service. The web service publishing, discovery and binding process is generally done by consumers at design time.

4.3 UDDI based mechanism : Reputation-Enhanced Web Service Discovery with QoS

The existing UDDI registries only support web services discovery based on the functionality of services. As the customers are interested in not only the functionalities of web services, but also their nonfunctional characteristics i.e. quality of service (QoS), that may have huge impact on the result of web service discovery. If there are multiple w eb services providing the same functionality in UDDI registries, the QoS requirement can be used as a finer search constraint. *Ziqiang Xu* et al [107] proposed a model of reputation-enhanced web services discovery with QoS to help consumers find the services that best meet their requirements.

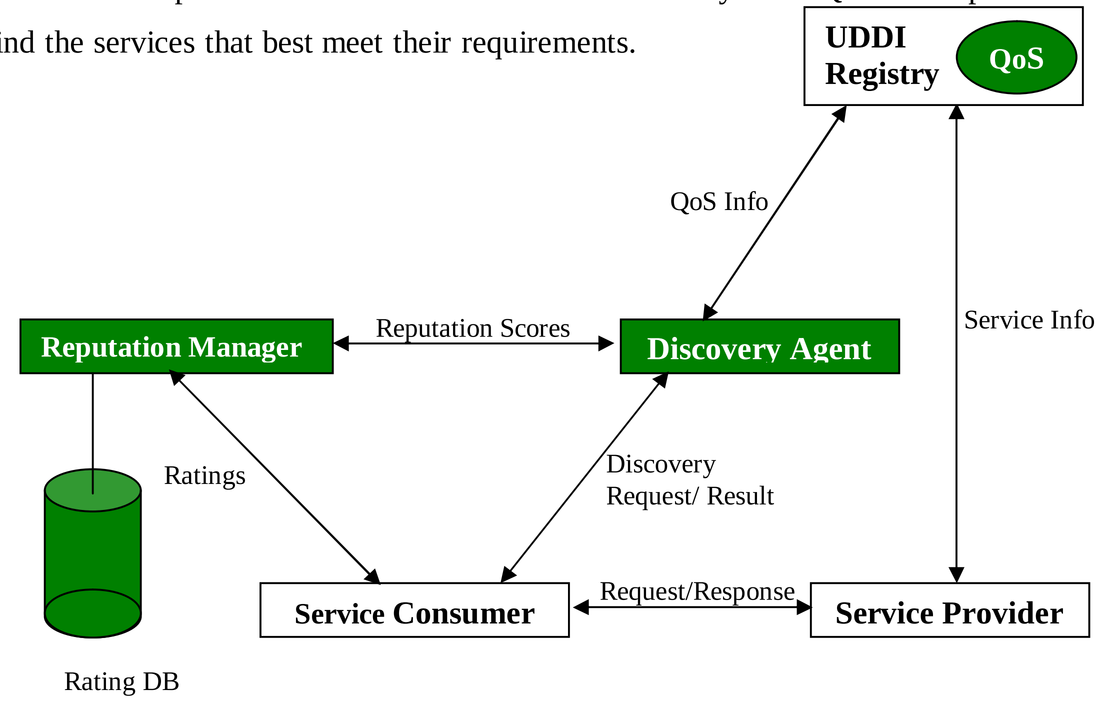


Figure 4.2 : Model of Reputation-enhanced Web Services Discovery with QoS

In this model, the UDDI registry is enhanced with QoS information, and two new roles, discovery agent and reputation manger, are added in our model as shown in Figure 4.2. The UDDI registry stores QoS information of services by using *tModels*. The discovery agent acts as a broker between a service consumer, a UDDI registry and a reputation manager to discover the web services that satisfy the consumer's functional, QoS and reputation requirements. The reputation manager collects and processes service ratings from consumers, and provides service reputation scores when requested by the discovery agent.

4.3.1 Publishing QoS Information

When a Web Service Provider publishes a web service, it creates and registers a *tModel* within a UDDI registry. The QoS information of the Web service is represented in the tModel, which is referenced in the binding template that represents the web service deployment. Each QoS attribute is represented by a *keyedReference* in the generated *tModel*. The name of a QoS attribute is specified by the *keyName*, and its value is specified by the *keyValue*. Instead of different units, default units are used for the QoS attributes values in the *tModel*. For example, the default unit used for price is CAN$ per transaction, for response time is second, for availability is percentage, and for throughput is transaction per second. For example a company publishes its Stock Quote service in a UDDI registry with the following QoS information:

- Service price: CAN $0.01 per transaction
- Average response time: 0.05 second
- Availability: 99.99%
- Throughput: 500 transaction/second

The company creates and registers a *tModel* that contains the QoS information for this service before it publishes the service with the UDDI registry. With QoS information of web services stored in *tModels* in a UDDI registry, service consumers can find the services that match their QoS requirements by querying the UDDI registry. The details of this process are discussed in the section 4.3.3.

```
<tModel tModelKey="somecompany.com:StockQuoteService: PrimaryBinding:QoSInformation"">
<name>QoS Information for Stock Quote Service</name>
<overviewDoc>
        <overviewURL>
                http://<URL describing schema of QoS attributes>
        <overviewURL>
<overviewDoc>
<categoryBag>
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:Price"
                keyName="Price Per Transaction"
                keyValue=" 0.01" />
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:ResponseTime"
                keyName="Average ResponseTime"
                keyValue="0.05" />
          <keyedReference
                tModelKey="uddi:uddi.org:QoS:Availability"
                keyName="Availability"
                keyValue="99.99" />
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:Throughput"
                keyName=" Throughput"
                keyValue="500" />
</categoryBag>
</tModel>
```

Figure 4.3 The *tModel* with the QoS information

4.3.2 Updating QoS Information

Web service Providers need to update the QoS information of their services in the UDDI registry frequently to ensure that the QoS information is accurate and up to date. Only a service provider that publishes a service and its QoS information in a UDDI registry can modify and update the QoS information. A service provider searches the UDDI registry to find the *tModel* that contains QoS information for the service it published before, updates the QoS information in the *tModel*, and then saves the *tModel* with the same *tModelKey* assigned previously.

4.3.3 Discovering web service through Discovery Agent and Reputation Manager

Service Consumer's request is received by a discovery agent first, it finds the services in the registry that match their requirements and then returns the response to the consumers. A request for web service discovery consists of functional, QoS, and reputation requirement of the web service. The format of request for web service discovery specifying the details of how to specify functional, QoS and reputation requirements is given in Figure 4.4. These types of SOAP messages for discovery requests are not generated manually by the service consumer, instead developers specify QoS and reputation requirements in a Java program that automatically generates required SOAP messages sent to the discovery agent. Customers can specify the following request parameters in the discovery request:

- Maximum number of services to be returned by the discovery agent
- Functional requirements: keywords in service name and description
- Service price: the maximum service price a customer is willing to pay
- Service performance and other QoS requirements such as response time, throughput, and availability.
- Dominant QoS attribute.
- Service reputation requirements.
- Weights for the QoS and reputation requirements

```
<?xml version="1.0" encoding="UTF-8" ?>
    <envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
        <body>
            <find_service generic="1.0" xmlns="urn:uddi-org:api">
                <functionalRequiremen>
                        Keywords in service name and description
                </functionalRequirement>
                <qualityRequirement weight=QoS Weight>
                        <dominantQoS>Dominant QoS</dominantQoS>
                        <QoS attribute 1>Value</QoS attribute 1>
                        <QoS attribute 2>Value</QoS attribute 2>
                        <QoS attribute 3>Value</QoS attribute 3>
                                  ……
                        <QoS attribute n>Value</ QoS attribute n>
                </qualityRequirement>
                <reputationRequirement weight=Reputation Weight>
                        <reputation>Reputation Score</reputation>
                        </reputationRequirement>
                <maxNumberService>Value</maxNumberService>
            </find_service>
         </body>
     </envelope>
```

Figure 4.4 Service Discovery Request Format

The dominant QoS attribute is the one that consumers consider as the most important and is used in the calculation of the QoS score for each service candidate in the service matching process. A consumer can specify QoS requirements only or both QoS and reputation requirements in the request. The weights for QoS and reputation requirements indicate their importance and they range from zero to one, where zero means no requirement for QoS or reputation while one means it is the only requirement on QoS or reputation. The sum of the weights must to one. Instead of setting the preference to each QoS attribute, a dominant QoS attribute is to be set having highest preference as it is easier for customers to sekect the most important QoS attribute than to specify separate priority for each of the QoS attributes. This will greatly simplify the calculation of QoS scores in the service ranking process.

As the discovery agent receives the request for service discovery, it find services in

UDDI registry that match the functional requirements specified as keywords in service name and also obtain QoS information of each stored in corresponding *tModels*. Then it matches the published QoS information with the QoS requirements specified in the request, finds the matched services, ranks the matches by QoS scores and/or reputation scores and returns the result to the customer. The QoS scores of services is calculated by the formula given as below :

$$\boldsymbol{QoSScore_i} = \begin{cases} \dfrac{DominantQoS_i}{BestDominantQoS} & \text{if dominant QoS attribute is monotonically increasing} \\[2ex] \dfrac{BestDominantQoS}{DominantQoS_i} & \text{if dominant QoS attribute is monotonically decreasing} \end{cases}$$

where $QoSScore_i$ is the QoS score of *ith* service,

i is the position of the service in the list of matched services,

$DominantQoS_i$ is the value of the dominant QoS attribute of service *i*, *BestDominantQoS* is the highest/lowest value of the dominant QoS attribute of the matched services when the dominant attribute is monotonically increasing/decreasing.

The adjusted Reputation scores of services is calculated by the formula given as below :

$$AdjRepuScore_i = \frac{RepuScore_i}{h}$$

where $AdjRepuScore_i$ is the adjusted reputation score of service *i*, *i* is the position of the service in the list of matched services,

$RepuScore_i$ is the original reputation score of service *i*,

h is the highest original reputation scores of the matched services.

The final overall scores of services required for ranking is given by the equation below:

$$\boxed{OverallScore_i = QoSScore_i \times QoSWeight + AdjRepuScore_i \times RepuWeight}$$

where $OverallScore_i$ is the overall score of service *i*,

i is the position of the service in the list of matched services,

$QoSScore_i$ is the QoS score of service *i*,

QoSWeight is the weight of QoS requirement specified by service consumers,

$AdjRepuScore_i$ is the adjusted reputation score of service *i*,

RepuWeight is the weight of reputation requirement specified by consumers.

A reputation manager in this service discovery model is based on the models proposed by *Majithia et al.* and *Wishart et al.* A QoS reputation score is calculated based on feedback by service consumers. Service Reputation Manager collects the data based on feedback from the service consumer, processes it and updates the reputation score. After using the web service, the service consumer rates it on a scale of 1 to 10 where, 10 means extreme satisfaction, 5 means average satisfaction and 1 means extreme dissatisfaction. Awarding bonus points to the consumers for their feedback will encourage them to provide valid ratings of the used services which can be used in service discovery to reduce the cost of the discovery.

The service rating storage is based on *SuperstringRep*, a protocol proposed by Wishart et al. The ratings of services by consumers are stored in the reputation manger's local database. Each rating record consists of service ID, consumer ID, rating value and a timestamp fields. The service key in the UDDI registry of the service is referred as the service ID, and the service consumer's IP address is used as the consumer ID.

Following table shows ratings records for services.

Table 4.2 Ratings record for services along with Timestamp

Service ID	Consumer ID	Rating	Timestamp
8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6	117.239.43.139	7	2011-09-03 10:15:34
8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6	172.50.43.30	8	2011-09-12 11:25:07
8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6	117.195.125.201	5	2011-09-11 19:20:12
53164900-f0b0-11d5-bca4-002035223h97	116.23.56.23	7	2011-09-21 12:15:02
8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6	117.239.43.137	5	2011-09-21 09:20:22
53164900-f0b0-11d5-bca4-002035223h97	172. 50.43.87	6	2011-10-29 09:20:22
b6cb1cf0-3aaf-11d5-80dc-002035245u62	117.239.43.131	7	2011-09-22 19:10:56
53164900-f0b0-11d5-bca4-002035223h97	117.239.43.134	6	2011-10-11 12:23:43
53164900-f0b0-11d5-bca4-002035223h97	116.23.56.26	8	2011-08-12 09:20:22

There are three services in the above table with Service ID “8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6”, “53164900-f0b0-11d5-bca4-002035223h97” and “b6cb1cf0-3aaf-11d5-80dc-002035245u62”, respectively. Each of the three services receives some ratings from consumers. Only one rating for a service per consumer is stored in the table. New ratings from the consumer for the same service replace older ratings. The timestamp is used to determine the latest rating for a particular service rating.

Reputation score for a web service is calculated on the basis of the work by *Majithia et al.* and the work by *Wishart et al.* *Majithia et al.* propose a method to calculate the reputation score as weighted sum of ratings for a service, where a coefficient is the weight attached to a particular context. *Wishart et al*. propose an aging function that applies a factor to each of the ratings regarding a service. In this model, the reputation score (*U*) of a service is calculated as the weighted average of all ratings the service received from customers, where an inclusion factor is the weight attached to each of the ratings for the service:

$$U = \frac{\sum_{i=1}^{N} S_i \gamma_i}{\sum_{i=1}^{N} \gamma_i}$$

$$\gamma_i = \lambda^{d_i}$$

where U is the reputation score for a service,

S_i is the ith service rating,

γ_i is the aging factor for ith service rating,

λ is the inclusion factor, $0 < \lambda < 1$,

d_i is the number of the days between the two times t_c and t_i:

t_c is the current time when the reputation score is computed,

t_i is the time of the ith rating for the service.

The inclusion factor λ is used to adjust the responsiveness of the reputation score to the changes in service activity. When λ is set to 0, all ratings, except the ones that are provided by consumers on the same day as the reputation score is computed, have a weight of 0 and are not be included in the computation. When λ is set to 1, all ratings have equal weight of 1 and used in the computation. A smaller λ means only recent ratings are included and a larger λ means more ratings are included.

Here, service matching, ranking and selection algorithm is based on the matching algorithm proposed by Maximilien and Singh. Simplified flowchart of improved algorithm is given below :

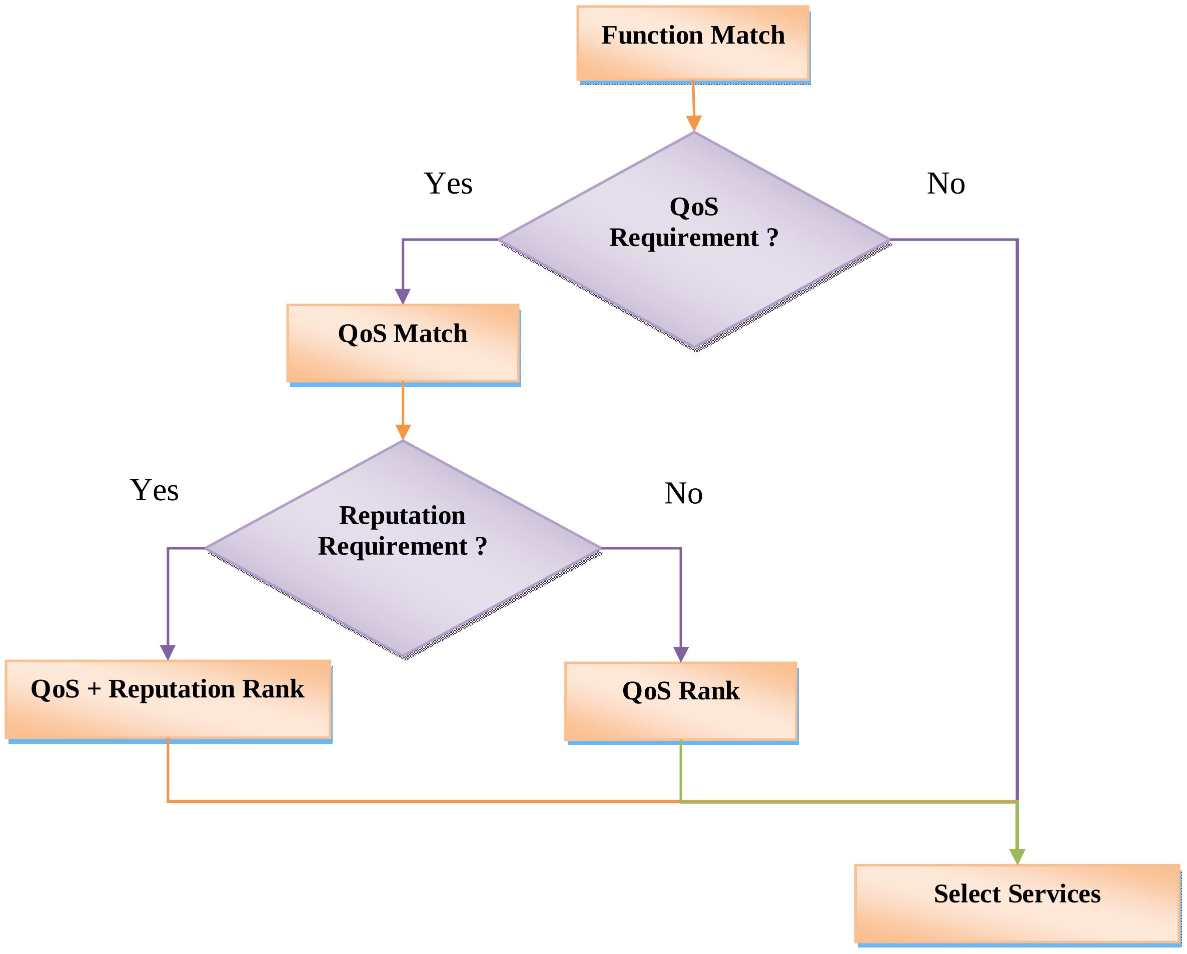


Figure 4.5 Flowchart for Matching, Ranking and Selecting service

Example :

SOAP Request for Web Service Discovery

```
<?xml version="1.0" encoding="UTF-8" ?>
      <envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
              <body>
                    <find_service generic="1.0" xmlns="urn:uddi-org:api">
                        <functionalRequirement>
                                Stock Quote
                        </functionalRequirement>
                        <qualityRequirement weight=0.4>
                                        <dominantQoS> availability</dominantQoS>
                                        <price>0.01</price>
                                        <responseTime>0.1</responseTime >
                                        <throughput>400</throughput>
                                        <availability>99.9</availability>
                        </qualityRequirement>
                        <reputationRequirement weight=0.6>
                                        <reputation>8</reputation>
                                        </reputationRequirement>
                        <maxNumberService>1</maxNumberService>
                    </find_service>
                </body>
        </envelope>
```

Figure 4.6 Service Discovery Request SOAP Message

SOAP Response for Web Service Discovery

On the request from service consumer, the discovery agent finds two services that match the requirements in the request, ranks the services using their QoS scores and reputation scores, and returns one service to the consumer since the request specifies the maximum number of services to be returned is 1. A SOAP message of service discovery response is shown in following Figure.

```
<?xml version="1.0" encoding="UTF-8" ?>
        <envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
                <body>
                    <serviceList generic="1.0" xmlns="urn:uddi-org:api" truncated="false">
                            <serviceInfos>
                                    <serviceInfo
                                    serviceKey="9521db61-eac1-42e4-5ab0-1d87b8f1876a"
                                    businessKey="8e4a1bc28-afb7-32a9-17ab-c2a32e6e1a27">
                                            <name>Stock Quote Canada</name>
                                            <qualityInformation>
                                                    <price>0.01</price>
                                                    <responseTime>0.08 </responseTime >
                                                    <throughput>800</throughput>
                                                    <availability>99.99</availability>
                                            </qualityInformation>
                                            <reputationInformation>9</reputationInformation>
                                      </serviceInfo>
                            </serviceInfos>
                    </serviceList>
                </body>
        </envelope>
```

Figure 4.7 Service Discovery Response SOAP Message

Summarizing above discussion on Reputation-Enhanced Web Services Discovery with QoS, it can be concluded that a reputation management system provides a mechanism to help a service discovery agent to improve the possibility to find services those match a consumer's functional, QoS and reputation requirement also provide consistently stable QoS performance. The problem of the accountability of those who provide ratings to the services still remain unsolved. In real world, ratings of a service could be provided by its competitors and trade partners or even the providers itself. Hence, assuming the service ratings are all trustworthy, service consumers could be easily misguided in case of service selection. A third party standard for ensuring the quality of service is needed.

4.4 UDDI based mechanism : Web service QoS-Certifier based Web Service Discovery

As we have discussed in earlier section, there is a need of some certifier agency who will certify the claims of quality of services or their ratings before publishing it to the UDDI registry to make it trustworthy. *Shuping Ran* [83] proposed framework that can serve to the service consumers needing quality of service assurance. There are four roles in this proposed model: Web service supplier, Web service consumer, Web service QoS certifier, and the new UDDI registry. As before, the Web service provider offers Web service by publishing the service into the registry, the Web service consumer needs the Web service offered by the provider, the new UDDI registry is a repository of registered Web services with lookup facilities and the new certifier's role is to verify service provider's QoS claims for publishing. The proposed new registry differs from the current UDDI model by having information about the functional description of the web service as well as its associated quality of service registered in the repository. Web service can be discovered by functional description of the desired web service, with the required quality of service attributes as requirement criteria. The new role in this model is the web service QoS certifier that does not exist in the original UDDI model. The certifier verifies the claims of quality of service for a web service before its registration.

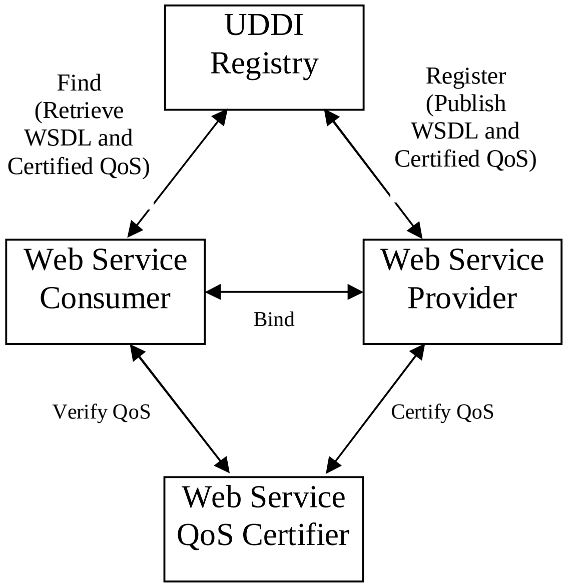

UDDI Registry
Find (Retrieve WSDL and Certified QoS)
Register (Publish WSDL and Certified QoS)
Web Service Consumer
Bind
Web Service Provider
Verify QoS
Certify QoS
Web Service QoS Certifier

Figure 4.8 Web services registration and discovery model with QoS Ceritifier

As shown in above figure, a web service provider supply service description along with its functional aspect as well as quality of service information related to the proposed web service. The claimed quality of service needs to be certified and registered in the repository. The web service provider first communicates its QoS claim to the web service QoS certifier before publishing in the UDDI registry. The certifier verifies the claims and either certifies or down grade the claim. The report is sent back to the service provider with certification identification information. This information is also registered in the certifier’s repository identified by a certification Id. The certifier provides a set of web services for any interested parties to access its repository about QoS claims for verification purposes. After the QoS certification been issued by the certifier, the supplier then registers with the UDDI registry with both functional description of the service and its associated certified quality of service information. The UDDI registry cross checks it with the certifier to ensure the existence of the certification. On successful checking, the registry then registers the service in its repository. In this framework, a new role is introduced– QoS Certifier who verifies the QoS claims from the web service suppliers and its role is very similar to rating agencies in other domains such as the financial sector, service industry etc, but the details regarding its implementation are unexplored yet.

4.5 jUDDI Registry working

jUDDI (pronounced "Judy") is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for (Web) Services. jUDDI is a pure Java web application and as such can be deployed to any application server or servlet engine that supports version 2.1 or later of the servlet API. jUDDI also requires an external datastore in which to persist the registry data it manages. Typically this is a relational database management system such as MySQL, Oracle or DB2. Support for several open source and commercial database products are included.

jUDDI consist of a core request processor that unmarshalls incoming UDDI requests, invoking the appropriate UDDI function and marshalling UDDI responses (marshalling and unmarshalling is the process of converting XML data to/from Java objects). To invoke a UDDI function, jUDDI employs the services of three configurable sub-components or modules that handle persistence (the DataStore), authentication (the Authenticator) and the generation of UUID's (the UUIDGen). jUDDI is bundled and pre-configured to use default implementations of each of these modules to help the registry up and running quickly. These sub-components are described briefly as below.

Persistence (jUDDI DataStore)

jUDDI needs a place to store it's registry data so it is understandable that jUDDI is pre-configured to use JDBC and any one of several different DBMSs to do this. The process of setting this up is simple. Start by creating a new jUDDI database using the instructions for the preferred DBMS, in my case I have used MySQL.

To complete the DataStore set up, it is required to configure a JNDI Datasource with a name of 'jdbc/juddiDB' in the application server, in my case I am using Apache Tomcat as a application server for deplyment. Datasource setup varies on an product-by-product basis.

Authentication (jUDDI Authenticator)

Authenticating a jUDDI publisher is a two-step process. The first step confirms that the ID/password combination provided by the user in a *get_authToken* request is valid. The default Authenticator implementation simply approves any authentication attempt. It is expected that a typical jUDDI deployment will use an external authentication mechanism. The second step confirms that the publisher has been defined to jUDDI. A publisher is said to be defined when a row identifying the publisher exists in the PUBLISHER table of the jUDDI datastore.

The PUBLISHER table consists of several columns but only four of them are required and they are defined as follows:

Table 4.3 Publisher table

Column Name	Description
PUBLISHER_ID	The user ID the publisher uses when authenticating.
PUBLISHER_NAME	The publisher's name.
ADMIN	Indicate if the publisher has administrative privileges. Valid values for this column are 'true' or 'false'.
ENABLED	Indicate if the publishers account is enabled and eligible for use.

The jUDDI web application will be extended to facilitate the Publisher creation process. The value of the ADMIN column in the PUBLISHER table above will be used to determine who has the privilege to create new jUDDI publishers.

UUID Generation (jUDDI UUIDGen)

The UDDI specification indicates that each Business, Service, Binding and TModel is to be uniquely identified by a Universally Unique ID (UUID). Additionally, jUDDI also uses the UUID generator to create AuthTokens.

Chapter 5

Smart Web service discovery enhanced with QoS Monitor

In Chapter 3, we have discussed two different web service registries, based on centralized approach, namely UDDI registry and ebXML registry which allow service providers to register their web services and service consumers to discover them. Though both of these registries are playing important role in e-business applications based on Service Oriented Architecture and have many things in common, in many aspects UDDI is architecturally superior to ebXML. Some more reasons to choose UDDI registry for service discovery are ebXML repositories are intended for more general purpose storage as compared to UDDI registry whereas UDDI is more focused on the kind of information that can be stored in the White, Yellow and Green pages; ebXML provide a global e-business standard of bigger size and magnitude that takes time and patience, which the industry either can't afford or chooses not to provide at this time whereas UDDI is not trying to own the world of e-business, but simply trying to facilitate all web-based services for query and introspection.

In Chapter 4, two different mechanisms based on UDDI registry are discussed in detail, which can be used to publish a web service along with its QoS information and discover a web service according the service consumer’s functional and QoS requirement. However, Reputation enhanced model for web service discovery model lacks trustworthiness of web service QoS claimed by service provider and QoS Certifier based mechanism just suggest the need of introducing a role of certifier in service oriented architecture, but lacks in implementation of the certifier. In this Chapter, we have discussed a new mechanism for web service discovery based on QoS which will rank services according to user’s preference of QoS and monitor ratings, at the same time ensuring those QoS values by monitoring them at regular intervals.

Even though many of previously discussed approaches also emphasize on web service discovery with QoS, none of them tackle the issue of trustworthiness of the published QoS information which is given by service provider themselves. Also there was no provision for specifying the priorities for each QoS parameter by the service consumer. In our proposed model, services whose QoS information is stored in the UDDI registry are monitored by Service Monitor on regular intervals and based on the deviation between published and actual QoS value, monitor ratings are given for each service. These monitor ratings are stored over the time period and used to calculate monitor score for each service. If the service consumer requests for a web service, he can specify his functional requirement, QoS requirement, domain requirement and monitor score requirement. Accordingly the Discovery agent will match, rank and select the services. Algorithms are proposed for service matching, ranking and selection which takes service monitoring into account in the ranking process.

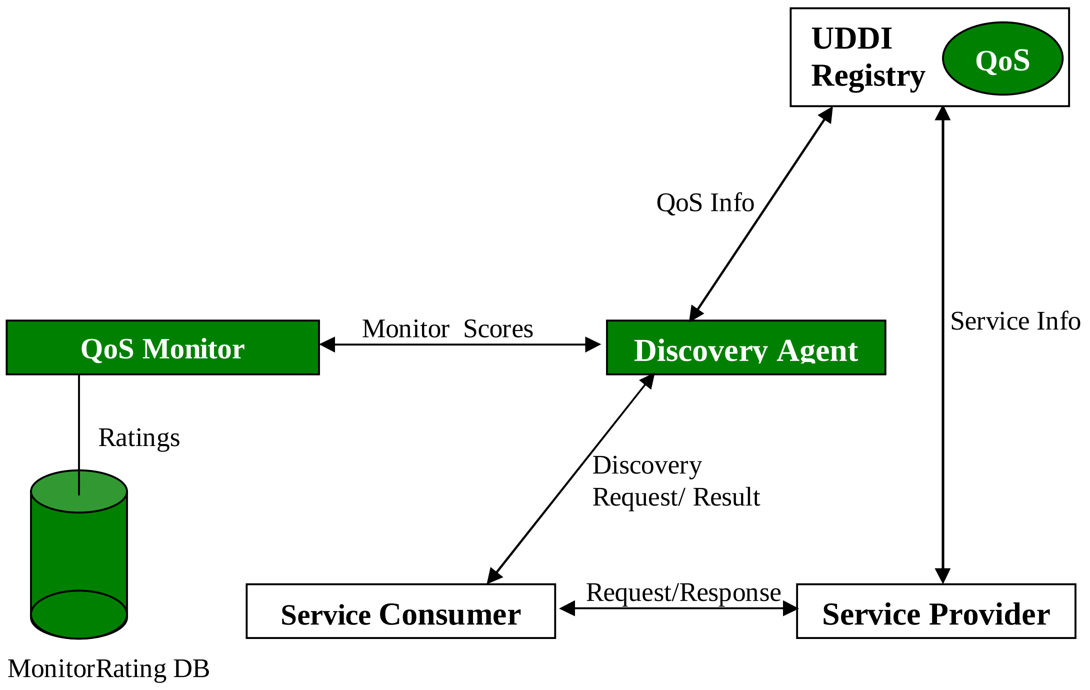


Figure 5.1 : Service QoS Monitor Based Model For Web Services Discovery

The standard SOA based models for web service publish and discovery comprise of three roles as service consumer who inquires for a web service, service provider who provides a web service and UDDI registry where information regarding web service is published.

In the proposed model, apart from these three roles, the UDDI registry is enhanced with QoS information, and two additional roles as a Discovery Agent and a Service Monitor as shown in Figure 5.1. QoS information of services provided by service consumer is stored in *tModels* in UDDI registry at the time of publishing a web service. The Discovery Agent assists service consumer to find the desired web service in the UDDI registry based on his service QoS requirement, domain requirement and monitor requirement with the help of a Service Monitor. The Service Monitor regularly (every week) monitors the services for verifying the QoS information provided by the service provider at the time of service publishing and updates monitor rating database. Based on those ratings, it provides service monitor scores to the discovery agent during web service discovery.

5.1 Publishing and updating QoS Parameters

QoS information of a web service is stored in one of the data structure of UDDI registry, *tModel*. When a service provider publishes a new web service in a UDDI registry, a *tModel* is created which stores the QoS information of the service and is registered with the registry. This *tModel* is referenced in the *bindingTemplate* that represents the web service deployment. Each QoS parameter is represented by a *keyedReference* in the generated *tModel*. The QoS parameter is specified by the *keyName*, and its value is specified by the *keyValue*. We assume default units for the values of QoS parameters and hence are not represented in the *tModel*.

<u>QoS Parameters</u>

The international quality standard ISO 8402 (part of the ISO 9000 (ISO9000 2002)) define quality as "*the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs*." We define quality of service as a set of non-functional characteristics that may affect the ability of the service to perform. QoS support for web services can provide a new business value to service providers by assuring a certain service quality for users. QoS parameters which we have discussed here are response time, reliability, availability, scalability and cost.

- Response time – The guaranteed max time required to complete a service request. In general, high quality web services should provide faster response time. The request time is the time when the client submits a request to the Web server, and the response time is the time when the server replies after processing the request. We have considered unit for measuring response time is *second*. It can be measured by keeping track of the timestamps at the service request time and service response times. If the time at which the client requests for web service is *t1* and the time at which the client receives response is *t2*, then the response time can be calculated as

Response time = *t2 – t1*

For Example, if the timestamp (*t1*) of client request is *15:25:00.812* and the timestamp (*t2*) of response to client is *15:25:01.968*, then the response time of web service can be calculated as *1.156 seconds*

- Throughput –Throughput is the number of requests completed over a period of time. Throughput can be measured by keeping track of the timestamps at the request time and response times. It is computed as the total number of requests divided by the elapsed time between the request time and the response time.

  Throughput = (number of requests processed)/(unit time)

  For Example, if the timestamp (*t1*) of client request is *16:09:00.324* and the timestamp (*t2*) of response to client is *16:09:00.350*, then its throughput can be calculated as 1/0.026 request per second.

  i.e. Throughput =~ 38 requests per second.

- Reliability – Reliability is the ability of a web service to perform its required functions under given conditions for a specified time interval. It also refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers. Reliability determines the percentage of the times an event is completed with success. This will help service consumer to expect the probability of a failure during a transaction. Service invocation attempt may either succeed or fail, and there is no middle way in that issue. Therefore, total number of service invocation attempts will be the number of failures added to the number of successful service invocations. We have measured reliability in %. Let *d* denote the number of days a web service is monitored for recording the number of failures. Let *n* be the number of failures encountered during that period. Here failures are considered as it is easy to count the number of failures than the successful service invocations. If *N* is the total number of events (number of successful service invocations plus number of failures), then the reliability or the success rate in one day can be derived.

Ratio of failure in *d* days = n / N

Daily average failure rate = $n / (N * t)$

Success rate or Reliability = $1 – (n /(N * t))$

For Example, if the service consumer sent requests to the web server for 5 days and the number of failures was counted and the reliability was calculated as follows:

Total number of service invocation attempts (number of successful service invocation plus number of failed events) in *d* days (N) = 520000 where *d* = 5 days.

Failures in *d* days (n) = 20000.

Failure rate $(n /(N * d))$ = 0.0077

Success rate or Reliability = 1 – 0.0077 = 0.9923

Hence, Reliability (%) = 99.23%

- Availability – Availability is the probability that the web service is up and in a readily consumable state. The web Service should be ready and available immediately when it is invoked. High availability ensures that the system failures or server failures would be less even during the peak times when there is heavy traffic to and from the server and that the given service is available continuously at all times. Let us say the “down time” is when a web service is not available. As the web service is either available or unavailable, the remaining time after subtracting the down time can be termed as the “up time” that means the web service is available.

Availability = 1 – [(down time)/(unit time period measured)]

If a web service *ws1* is monitored over 1 week i.e. *7 days x 24 hours x 60 minutes x 60 seconds = 604800 seconds*, we found it was down for 2 hours i.e. *2 hours x 60 minutes x 60 seconds = 7200 seconds* in the whole week. The availability of web service ws1 can be computed as –

Availibility = 1 – (7200/604800) = 0.9881

Availibility (%) of *ws1* is 98.81%

- Cost – Cost is the measure of the cost of requesting a service, which is specified by service provider at the time of publishing a service. It may be charged per service requests, or could be a flat rate charged for a period of time. For Example cost of *ws1* is $10 per year.

For example, a company XYZ publishes its Currency Converter service in a UDDI registry with the following QoS information:

• Response time:	0.10 second
• Throughput:	250 transaction/second
• Reliability:	99.9%
• Availability:	99.9%
• Cost:	USD 0.01 per transaction

```
<tModel tModelKey="xyz.com:CurrencyConverterService: PrimaryBinding:QoSInformation"">
<name>QoS Information for CurrencyConverterService </name>
<overviewDoc>
        <overviewURL>
                http://xyz.com/qos.xsd
        <overviewURL>
<overviewDoc>
<categoryBag>
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:ResponseTime"
                keyName="ResponseTime"
                keyValue="0.10" />
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:Throughput"
                keyName=" Throughput"
                keyValue="250" />
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:Reliability"
                keyName="Availability"
                keyValue="99.9" />
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:Availability"
                keyName="Availability"
                keyValue="99.9" />
        <keyedReference
                tModelKey="uddi:uddi.org:QoS:Cost"
                keyName="Cost"
                keyValue=" 0.01" />

</categoryBag>
</tModel>
```

Figure 5.2 The *tModel* with the QoS information

The company XYZ creates and registers a *tModel* that contains the QoS information for this service before it publishes the service with the UDDI registry. An Application Programming Interface (API) to the UDDI registry, such as UDDI4J [34], may be used to facilitate the service publishing process. Above Figure shows an example of this *tModel*. Service consumers can find the desired web service that match their QoS requirement with its QoS information stored in *tModels* as shown above in a UDDI registry, by querying the UDDI registry.

If QoS information of a web service stored in *tModel* in UDDI registry need to be updated, only service provider has the right to modify and update it. To facilitate the process of updating QoS information, an API to the UDDI registry, such as UDDI4J may be used. At the time of updating service QoS information, first it retrieves the registered *tModel* for the service from the UDDI registry, updates its content and saves it with the same *tModelkey*.

5.1.1 Algorithm for Publishing Service in UDDI Registry

When a new service provider wants to publish his services in the UDDI registry, he has to first register with the registry. After registration a user id and password is assigned to him, with which they will create a *businessEntity* and save it in the registry. Under that *businessEntity*, a web service is published along with its QoS information according the steps shown in algorithm in Figure 5.3

```
Algorithm 5.2.1: Publishing Web Service
Input : authToken, businessInfo, serviceInfo, QoS of service, domain
Output : Published services for a.business
Method :
1. Accept an authToken by passing user id and password registered at the
   UDDI registry
2. Create a business entity to represent the provider
3. For each service to be published
       (i)   Create a tModel to represent the QoS information and domain
             for the service, save it in the UDDI registry
       (ii)  Create a bindingTemplate containing a reference to the tModel
       (iii) Create a service entity to represent the service that the provider
             is publishing
       (iv)  Set the reference to the bindingTemplate in the service entity
       (v)   Add the service entity to the business entity
4. Save the business entity in the UDDI registry, receive a businessKey and
   a list of service keys assigned by the UDDI registry
```

Figure 5.3 Algorithm of Publishing Web Service

5.1.2 Algorithm for Updating Service QoS information in UDDI Registry

Over the time period, if the service provider need to update QoS information for a particular web service published in the UDD registry, he can retrieve the business entities and service entities by providing the *businessKey* and *serviceKey*. Figure 5.4 shows a detailed algorithm for service QoS information update process.

```
Algorithm 5.2.1: Updating Service QoS Information
Input : authToken, businessKey, serviceKey, new QoS of service
Output : Updated service with new QoS information
Method :
1. Accept an authToken by passing user id and password registered at the
   UDDI registry.
2. Find the business entity representing the provider with businesskey.
3. For each service to be updated
      (i)    Find the service entity representing the service that is to be
             updated with the servicekey.
      (ii)   Find and update the tModel representing the QoS information
             for the service with new QoS information, save it in the UDDI
             registry with the same tModelkey.
```

Figure 5.4 Algorithm of Updating Service QoS Information

5.2 Discovering Service in UDDI Registry

A service consumer sends a request for a web service to the registry in which he specifies the functional requirement, QoS and/or monitoring requirement. As a discovery agent receives the request, it matches the service requirements with the registered services, ranks the matched services according to the QoS and/or monitoring requirement specified and returns them to the consumers. Figure 5.5 shows the detailed algorithm of how discovery agent finds services that meet a consumer’s functional, QoS and/or monitoring requirements

```
Algorithm 5.2.1: Discovering a Web Service
Input : authToken, functionalReq, QoSReq, domainReq
Output : Published services for a.business
Method :
1. Accept an authToken by passing user id and password registered at the UDDI
   registry
2. Find services that match the customer’s functional requirements.
3. For each of the services that meet the customer’s functional requirements
      (i)    Find the service entity representing service in the UDDI registry with
             the serviceKey.
      (vi)   Find the tModel representing the QoS information and domain for the
             service.
      (vii)  Add the serviceKey to the short listed service list if the service’s QoS
             information in the tModel meets the customer’s QoS requirements
             and domain match the customer’s domain specification.
4. Determine ranking of each service in the short listed list based on their QoS
   score and Monitor score, arrange them in ascending order of their rank and
   return the specified number of services based on the consumer’s requirement
```

Figure 5.5 Algorithm of Discovering a Web Service

For example, consider a company looking for a Currency Converter service for its business application. The company specify the following service requirement details as shown in Table 5.1 :

Service name or description : Currency Converter		
QoS Parameter	QoS Value	Preference Order
Response time:	0.05 second	1
Throughput:	500 trans/sec	4
Reliability:	99.0%	2
Availability:	99.0%	3
Cost:	USD 0.02 per trans	5
Monitor Score : > 9.0		

Table 5.1 : Service requirements – functional, QoS and monitoring requirement

The company relies more on monitor score (more QoS assured service) than on QoS of the service, so it specifies a weight of 0.6 to the monitor requirement and a weight of 0.4 to the QoS requirement in the discovery request. Preference order for each QoS is also specified in the request as shown in Table 5.1. Figure 5.6 shows a SOAP request example for service discovery with these QoS and Monitor requirements.

```
<?xml version="1.0" encoding="UTF-8" ?>
    <envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
          <body>
               <find_service generic="1.0" xmlns="urn:uddi-org:api">
                    <functionalRequirement>
                         Currency Converter
                    </functionalRequirement>
                    <qualityRequirement weight=0.4>
                         <responseTime pref = “1”>0.05</responseTime >
                         <throughput pref = “4”>500</throughput>
                         <reliability pref = “2”>99.0</ reliability >
                         <availability pref = “3”>99.0</availability>
                         <price pref = “5”>0.02</price>
                    </qualityRequirement>
                    <monitorRequirement weight=0.6>
                         <monitorScore>8</monitorScore >
                    </monitorRequirement>
                    <maxNumberService>3</maxNumberService>
               </find_service>
          </body>
     </envelope>
```

Figure 5.6 : Service Discovery Request SOAP Message

```
<?xml version="1.0" encoding="UTF-8" ?>
        <envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
                <body>
                    <serviceList generic="1.0" xmlns="urn:uddi-org:api" truncated="false">
                            <serviceInfos>
                                    <serviceInfo
                                    serviceKey="9521db61-eac1-42e4-5ab0-1d87b8f1876a"
                                    businessKey="8e4a1bc28-afb7-32a9-17ab-c2a32e6e1a27">
                                            <name>Currency Converter1</name>
                                            <qualityInformation>
                                                    <responseTime>0.05</responseTime >
                                                    <throughput>400</throughput>
                                                    <reliability>98.0</ reliability >
                                                    <availability>97.0</availability>
                                                    <price>0.03</price>
                                            </qualityInformation>
                                            <monitorScore>9</monitorScore>
                                      </serviceInfo>
                                    <serviceInfo
                                    serviceKey="9643cb23-bca3-51f4-6ab1-2b76f3e5215b"
                                    businessKey="7d3b2cd54-abc8-43b6-24cf-d3c13e7e2b16">
                                            <name>Currency Converter2</name>
                                            <qualityInformation>
                                                    <responseTime>0.06</responseTime >
                                                    <throughput>350</throughput>
                                                    <reliability>97.0</ reliability >
                                                    <availability>98.0</availability>
                                                    <price>0.02</price>
                                            </qualityInformation>
                                            <monitorScore>9</monitorScore>
                                      </serviceInfo>
                                    <serviceInfo
                                    serviceKey="7221bd32-abc2-12d5-2cb3-4c01b4d1876a"
                                    businessKey="6f4b2bb64-afc4-22b3-24ac-b3a24e5e1a18>
                                            <name>Currency Converer3 </name>
                                            <qualityInformation>
                                                    <responseTime>0.07</responseTime >
                                                    <throughput>350</throughput>
                                                    <reliability>97.0</ reliability >
                                                    <availability>97.0</availability>
                                                    <price>0.02</price>
                                            </qualityInformation>
                                            <monitorScore>9</monitorScore>
                                      </serviceInfo>
                            </serviceInfos>
                    </serviceList>
                </body>
        </envelope>
```

Figure 5.7 : Service Discovery Response SOAP Message

When this SOAP request from service consumer is received by the discovery agent, it finds 10 web services that meet the requirements specified in the request, ranks them using their QoS scores and monitor scores and returns 3 top ranking services to the consumer as the request specifies the maximum number of services to be returned are 3. The discovery agent generates a SOAP message response of service discovery as shown in Figure 5.7

5.2.1 Discovery Agent Workflow

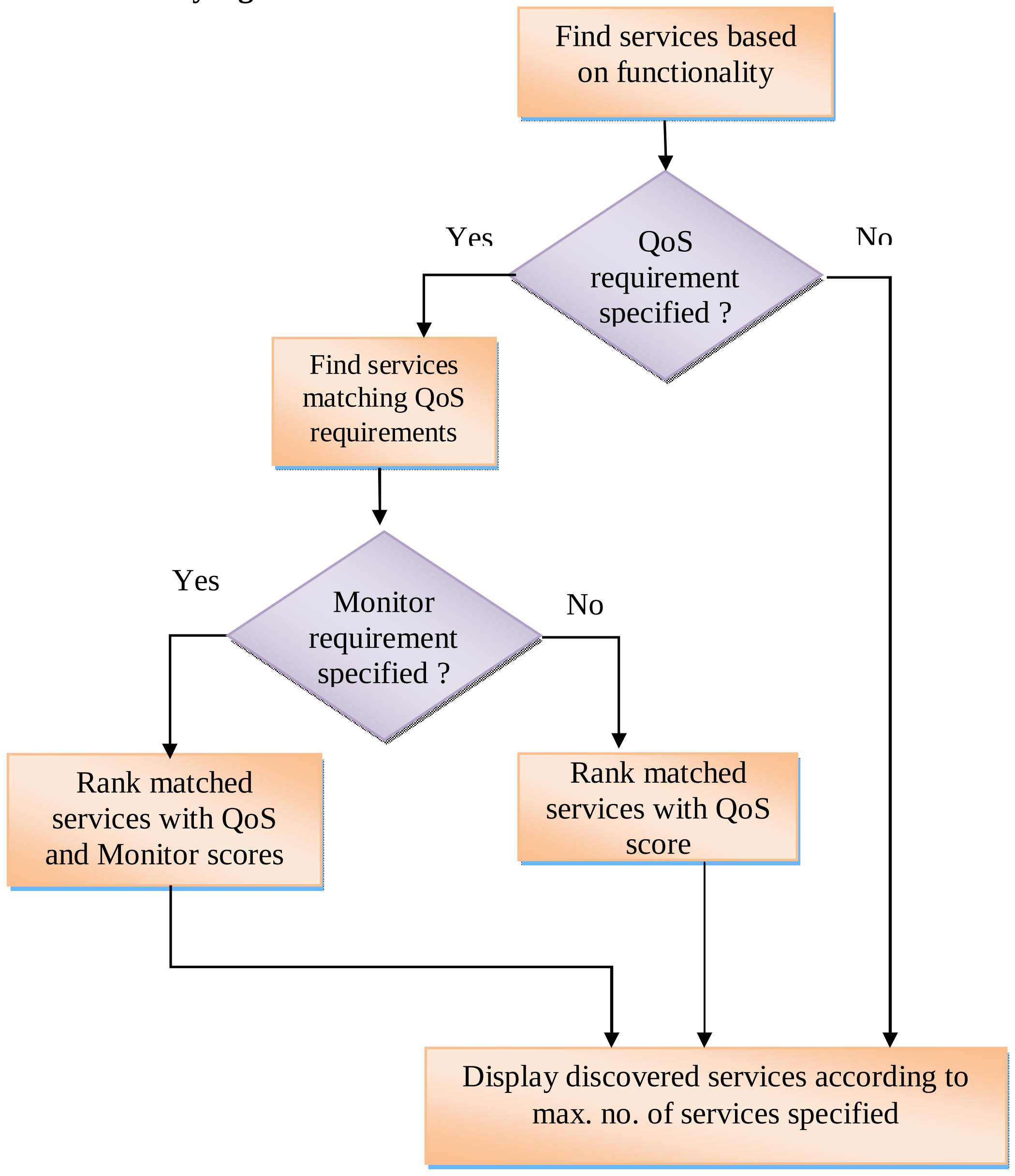

Find services based on functionality
QoS requirement specified ?
Yes
No
Find services matching QoS requirements
Monitor requirement specified ?
Yes
No
Rank matched services with QoS and Monitor scores
Rank matched services with QoS score
Display discovered services according to max. no. of services specified

Figure 5.8 : Discovery Agent Workflow for Matching, Ranking and Selecting service

Figure 5.8 gives an idea of the high level view of the discovery agent workflow and Figure 5.9 shows the steps of an overall algorithm for discovering the services.

Algorithm 5.3.1: Overall algorithm for finding Services
Input : functReq, qosReqt, MonitorReq, maxNumServices
Output : Ranked services
Method :

```
findServices (functReq, qosReqt, domain, MonitorReq, maxNumServices)
{
    // find services that match the functional requirements
    funct_Matched = funct_Match(functReq);

    if QoS requirements are specified in request
            // find services that match QoS requirements
            qosMatches = qosMatch (funct_Matched, qosReqt);

    else
            // return services according to the maxNumServices to be returned
            return selectServices(funct_Match, maxNumServices, “RANDOM”);

    if MonitorReq specified
            // rank matched services with QoS and Monitor scores
            shortlisted = monitorRank(qosMatches, qosReqt, MonitorReq);
            /* return services according to the maxNumServices to be returned based
            on overall score */
            return selectServices(shortlisted, maxNumServices,
    “OVERALL_SCORE”);

    else
            // rank matched services with QoS score
            shortlisted = qosRank(qosMatches, qosReqt);
            /* return services according to the maxNumServices to be returned based
            on QoS score */
            return selectServices(shortlisted, maxNumServices, “QOS_SCORE”);
}
```

Figure 5.9 : Overall Algorithm for service discovery

In the above algorithm,

funct_Match returns a list of services that satisfy the functional requirement.

qosMatch returns the services that satisfy the QoS requirements.

qosRank calculates QoS scores of all the services returned by the method *qosMatch* and returns a list of services sorted by the QoS score in descending order.

monitorRank calculates monitor score of all the services returned by the method q*osMatch*, removes services whose monitor scores are below the monitor requirement, calculates overall scores for remaining services and returns a list of services sorted by the overall score in descending order.

selectServices returns a list of services according to the maximum number of services to be returned in the discovery request.

5.2.1.1 Service QoS Matching

In an algorithm given in Figure 5.10, a list of functionally matched services *funct_Matched* and QoS requirements *qosReqt* are specified by service consumer are given as input. For every services in the functionally matched service list, QoS requirements specified by the consumer are matched with the published QoS information and if matched, then those QoS matched services are returned as *qos_matched.*

```
Algorithm 5.3.2: Find services that match QoS requirements
Input : funct_Matched, qosReqt
Output : Matched services
Method :

1. Initialize qos_matched to empty list.
2. For each service s in funct_Matched service list
      (i)   Obtain QoS information qosPub from UDDI registry
      (ii)  If qosPub is available and match with qosReq, then add service s in
            qos_matched list.

3. Return service list qos_matched.
```

Figure 5.10 : Service QoS Matching Algorithm

5.2.1.2 Service QoS Ranking

In an algorithm given in Figure 5.11, QoS matched services *qos_matched,* QoS requirement vales *qosReqt* and preference order *pref_val* of each QoS specified by service consumer are given as input. Assuming the default units for QoS parameter values, first its type is checked as whether a particular QoS is mono increasing or mono decreasing. Based on the type, QoS score is calculated for each service. In calculation of QoS score, preference order *pref_val* of each QoS parameter is also considered. Then services are sorted in descending order of QoS score and these QoS ranked services are returned.

```
Algorithm 5.3.3: Rank matched services by QoS information
Input : qos_matched, qosReqt, pref_val
Output : QoS Ranked services
Method :

1. For each service s in qos_matched list
    (i)    For each QoS parameter in qosReqt,

           a) Find the highest QoS value bestQoSVal
           b) If qosReqt.QoS.type is monoIncreasing, then calculate QoS_Score
              of each service as
              s.QoS_Score = sum(qosReqt.QoS.value / bestQoSVal) * pref_val
              Else
              s.QoS_Score = sum(bestQoSVal  / qosReqt.QoS.value ) * pref_val

2. Sort the service list on the basis of calculated QoS_Score in descending order.
3. Return the sorted service list.
```

Figure 5.11 : Service QoS Ranking algorithm

5.2.2 Monitoring Service QoS

Web Services are designed, developed and used like any other typical software. However, various roles are involved during each phase of life cycle of a web service. In a web service publishing/registration phase, service provider publishes service description as well as QoS information in UDDI registry. A service consumer finds the desired services by specifying functionality i.e. keyword and preferences of QoS parameters retrieved from the UDDI registry. During this process, a service consumer also expects service quality assurance. QoS Monitor component of Smart WebService Discovery system continuously monitors web service qualities and based on the deviation found between monitored and published QoS information, gives monitor ratings to each registered service. These monitor ratings are used for calculating monitor score needed to rank the services during service discovery process.

If the service consumer specifies a monitoring requirement in the service discovery request, the discovery agent removes those services from the matched service list whose monitor score is either unavailable or below the specified requirement. During this if only one service remains, without processing further, it is returned to the consumer as it is the only service that meets the consumer's QoS and monitor requirement. If there are more that one services meeting the consumer's QoS and monitor requirement, QoS scores are calculated for each as described in earlier section. Monitor scores of those matched services are then adjusted using a factor *f* so that adjusted monitor scores range from 0.1 to 1. Factor *f* is calculated as $f = 1 / h$, where *h* is the highest monitor score in the matched service list. Then all monitor scores are multiplied by the factor *f* so that the the score of the service with best monitoring result is adjusted to 1, and the other services' scores are adjusted based on their original monitor scores. At the end, the discovery agent calculates an overall score, which is as weighted sum of the QoS score and the adjusted monitor score, for all services based on the weights of the QoS and monitor requirements specified by the customer in the discovery request. Then a number of services are selected according to the maximum number of services(N) to be returned in the request. If N is greater than 1, the top N services with the highest overall scores are returned to the consumer else one service is randomly selected from those whose overall score is greater than LowLimit.

Figure 5.12 shows the steps for finding out those services from the QoS matched service list, whose monitor rating is available and meed the consumer monitor requirement.

```
Algorithm 5.3.4: Find QoS matched services those also match Monitor requirements
Input : qos_matched, qosReqt, MonitorReq
Output : Monitor Ranked services
Method :

1. For each service s in qos_matched list
      (i)     Obtain Monitor_rating from Monitor
      (ii)    If Monitor_rating is available and above MonitorReq value, then add
              service s in monitor_matched list
      (iii)   Else remove service s from the list.

2. Return the service list monitor_matched list.
```

Figure 5.12 : Service QoS and Monitor Matching Algorithm

Monitor ratings are calculated according the algorithm given in Figure 5.13. This algorithm will be auto executed weekly for monitoring the services registered in UDDI. A test client application for measuring real time QoS will be invoking all the registered web services on every week. The result of it containing monitored values of QoS along with the timestamp on which the monitoring was done is recorded in Monitor Rating database. Based on the deviations found between published QoS values and monitored QoS values during monitoring services, monitor ratings will be calculated and stored in the database which will be used for calculating monitor score of services.

Algorithm 5.3.5: Calculate Monitor rating for each service
Input : services, qosPub, qosMon
Output : Services with Monitor rating for each service
Method :

```
1. For each service s in services list
      (i)   for each qos parameter, compute qosMon as
              a) for each timestamp i from 1 to n (no. of timestamps ie. weeks)

                 qosMon = sum_qos / n

              b) Compute deviation factor as,

                 Δqos_diff = qosPub – qosMon

              c) If  Δqos_diff <= 5 then
                         rating = 10
                 Else if Δqos_diff <= 10 then
                         rating = 9
                 Else if Δqos_diff <= 15 then
                         rating = 8
                 Else if Δqos_diff <= 20 then
                         rating = 7
                 Else if Δqos_diff <= 25 then
                         rating = 6
                 Else if Δqos_diff <= 30 then
                         rating = 5
                 Else if Δqos_diff <= 35 then
                         rating = 4
                 Else if Δqos_diff <= 40 then
                         rating = 3
                 Else if Δqos_diff <= 45 then
                         rating = 2
                 Else if Δqos_diff <= 50 then
                         rating = 1
                 Else
                         rating = 0
      (ii)  Compute final Monitor_rating as,

            Monitor_rating = sum of all ratings / number of qos parameters

2. Return the service list services with Monitor_rating.
```

Figure 5.13 : Monitor Rating Algorithm

In this algorithm, the monitor score of a service is computed as the weighted average of all monitor ratings the service received from service monitor, where an inclusion factor is the weight attached to each of the ratings for the service. The inclusion factor λ ($0 < \lambda < 1$) is used to adjust the responsiveness of the monitor score to the changes in service activity. When λ is set to 0, all ratings, except the ones that are provided by monitor on the same day as the monitor score is computed, have a weight of 0 and are not be included in the computation. When λ is set to 1, all ratings have equal weight of 1 and used in the computation. A smaller λ means only recent ratings are included and a larger λ means more ratings are included. Figure 5.14 shows an algorithm to calculate monitor score for all services and returns those services for further processing.

```
Algorithm 5.3.6: Calculate Monitor score for each service
Input : services, S (Service Rating), Rating Time
Output : Services with monitor score
Method :

1. For each service s in services list
     (i)   Initialize sum_ratings, sum_aging to 0.
     (ii)  For each rating i to n
             a) Calculate aging factor as
                aging factor ($\gamma_i$) = $\lambda^{d_i}$
                     where $d_i$ = the number of the days between the
                           current time when the monitor score is
                           computed and the time of the ith rating for
                           the service
             b) $sum\_ratings = sum\_ratings + S_i * \gamma_i$
             c) $sum\_aging = sum\_aging + \gamma_i$
     (iii) Compute monitorScore as
                $monitorScore = sum\_ratings / sum\_aging$
2. Return the service list services with monitorScore.
```

Figure 5.14 : Monitor Score Calculation Algorithm

Figure 5.15 shows the algorithm for calculating adjusted monitor score which will be used for calculating overall score of each service.

```
Algorithm 5.3.7: Calculate adjusted Monitor score for each service and rank them
Input : services
Output : Ranked service List with adjusted Monitor score for each service
Method :

1. Find the highest monitor score bestMonScore from the service list servicse.
2. For each service s in services list
       Compute s.adj_monitorScore = s.monitorScore / bestMonScore

3. Sort the service list on the basis of calculated monitorScore  in descending order
4. Return the service list services.
```

Figure 5.15 : Adjusted Monitor Score Calculation Algorithm

In the algorithm shown in Figure 5.16, overall score for each service is calculated with QoS Score, adjusted monitor score (obtained from above algorithms) and QoS weight, Monitor weight requirement specified in service consumer’s request.

```
Algorithm 5.3.8: Calculate overall score for each service
Input : services, qosWeight, monWeight
Output : Adjusted Monitor score for each service
Method : Service List with overall score for each service

1. For each service s in services list
       Compute s.overallScore as
       s.overallScore  = s.QoSScore * qosWeight + s.adj_monitorScore * monWeight

2. Return the service list services.
```

Figure 5.16 : Overall Score Calculation Algorithm

Figure 5.17 shows algorithm for service selection according to the maximum number of services returned.

Algorithm 5.3.9: Select services according to the maximum number of services returned

Input : services, option, maxNumServices

Output : Ranked services

Method :

```
1. Initialize final_servicelist to empty list of services.
2. If maxNumServices > 1, then
     i.   Initialize count to 0.
     ii.  while (count < maxNumServices and count < qos_matched.size())
              a) final_servicelist.add(services[count])
              b) Increment count by 1.
3. Else
     i.   Initialize candidate_services to empty list of services.
     ii.  If option = "RANDOM" then,
            candidate_services = services
     iii. Else
              a) For each s in services
                 If option = "QOS_SCORE" then
                        If s.QoS_Score >= LowLimit, then add service s to
                        candidate_services
                 Else
                        If s.overallScore >= LowLimit, then add service s to
                        candidate_services


                 End For
              b) rnum = random(0, candidate_services.size())
              c) final_servicelist.add(services[rnum])
4. Return final_servicelist.
```

Figure 5.17 : Service Selection Algorithm

Chapter 6

Experiments and Results Analysis

In this chapter, we describe the experimental setup and implementation of the proposed model for service discovery in the first section. In the next section, the results of the evaluation of the experiments performed are presented. To demonstrate the working of the model, we have illustrated two scenarios as : service providers publishing services with QoS information in the UDDI registry and service consumers discovering services that meet their functional and QoS requirements through discovery agent and service monitor. A set of experiments is performed to evaluate our service matching, ranking and selection algorithm and discussed the experimental results. Ultimately, the objective of this evaluation is to show that by using the proposed model based on the algorithms stated in Chapter 5, there are higher chances of selecting the most appropriate web services with the desired and assured QoS for the consumer than those that do not meet these requirements.

The implementation of service discovery model and the results of two scenarios are described in section 6.1.The experimental setup for evaluating the proposed algorithms and their results are presented in section 6.2.

6.1 Service Discovery Model implementation

The Service discovery model is implemented with the three components running on three separate machines, as shown in Figure 6.1:

UDDI Registry and Discovery Agent : We have used jUDDI (Version 2.0rc5 based on UDDI 2.0) to set up our own UDDI registry on the one machine, which is connected to a local MySQL (Version 5.0.0) database. Our registry jUDDI and discovery agent program based on the proposed algorithms run on TomCat 6.0 on the same server.

Service Consumer: A Java Program simulated as a service consumer run on second machine. This program can send a service discovery request to the discovery agent program running on the registry server to find services that meet its requirements.

Service Providers: A Java Program simulated as a service provider run on third machine. This program publishes the web service, its QoS and updates it.

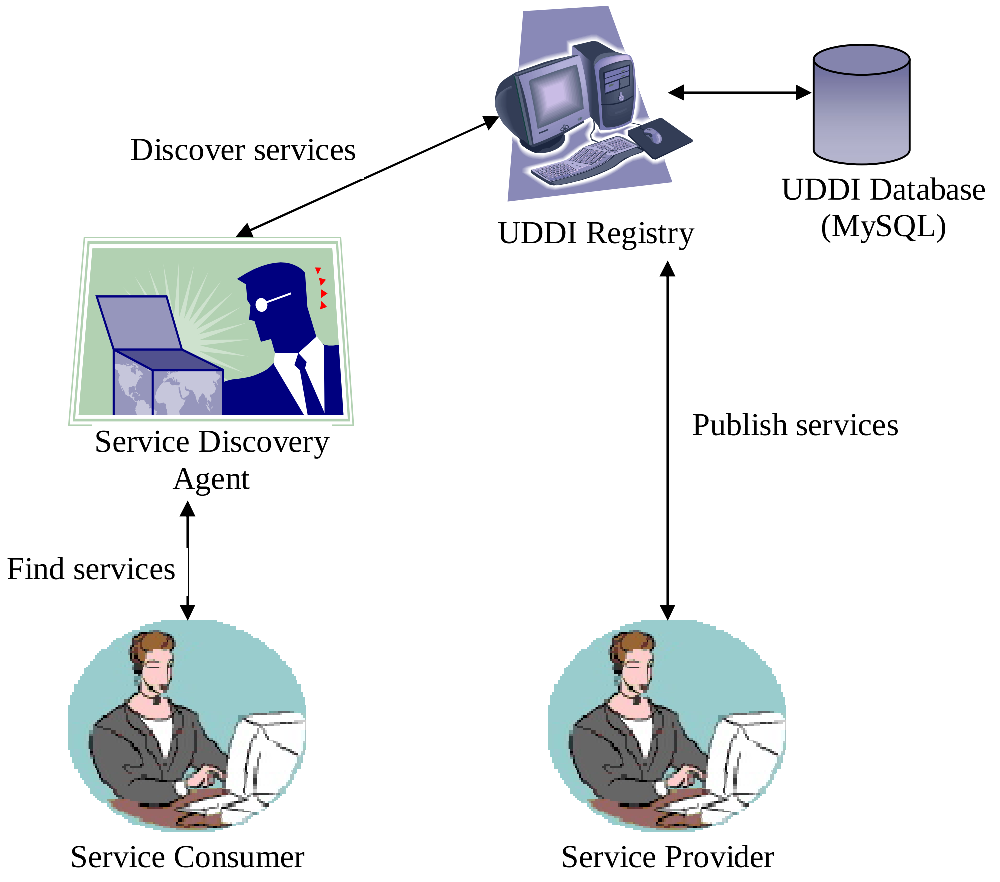


Figure 6.1 : Service Discovery Model

The three machines we used in our experiments are HP Compaq PC with Windows XP operating system. The configuration of each PC is 1.60 GHz Intel Pentium 4 processor and 2 GB RAM.

<u>Scenario 1: Web Service Publishing with QoS</u>

In this scenario, we have demonstrated about how the service providers publish their web services with QoS information in the UDDI registry. There are five different service providers who publish their individual Currency Converter web service in the UDDI registry with different QoS values. After executing the service provider program for publishing service, another program to find the published service and its QoS tModel, is executed using the service key assigned by the UDDI registry during publishing process. The QoS tModel for the given service key is found and its contents are checked.

Scenario 2: Web Service Discovery

In this scenario, the demonstration of how service consumer finds services that meet their functional and assured QoS requirements though a discovery agent. In the experiment, the service consumer looks for Currency Converter web service whose response time is 0.02 sec, availability is 98%, reliability is 99%, throughput is 400 trans/sec and price is $0.1. The preference order of QoS required for the web service is as response time 1st, reliability 2nd, availability 3rd, throughput 4th, and price 5th. This means faster response is the highest priority comparing to the price of the web service. When a service consumer program is executed, a service discovery request is send to the discovery agent program. The agent inquires the UDDI registry to find the services those meet the consumer's functional requirements (ie. Currency Converter), retrieve the QoS *tModel* for each of the matched services and checks if the published QoS parameter values in the tModel matches the required QoS in the discovery request. For each of the matched services, monitor score will be calculated based on the monitor ratings obtained from the monitor over the period of 1 month. If the live monitor score matches with the monitor score specified in the request, those services are short listed for the selection. Then, maximum number of services to be returned, as specified in the request are selected from that list and send in the response to the service consumer.

6.2 Experimental Setup and Evaluation

We have described experimental setup and execution of discovery request in this section. In the end, a set of experiments designed to evaluate the proposed algorithms and results of experiments are presented. From the experimental results we have shown that the chances of selecting a service that best match the consumer’s requirements are increased by using the algorithms. We showed that the service those show higher deviation between published QoS and actual QoS, that means whose monitor score is less, are less likely to be selected than those services with higher monitor score. We also showed that based on the QoS score and Monitor score weightage specified by the consumer in the requirement, the service selection is affected. An observation of selecting a suitable inclusion factor for reducing inconsistency in monitor score is also noted.

The experimental setup for the proposed algorithms is shown in Figure 6.2. Service consumers send service discovery requests with different QoS and monitor requirements. When the discovery agent program receives the request from the consumer, it retrieves the QoS information from the database and executes the matching algorithm. If the consumer specifies a monitoring requirement in the request, the agent program communicates with service monitor program, which calculates and returns the monitor score to the agent. The discovery agent ranks the matched services based on their QoS and monitor scores, selects services that best meets the consumer’s requirements and returns them to the consumer. In the following experiments, we have considered all services having the same functionalities. All the consumers request the same functional requirements which are satisfied by these services. The QoS information of all these services varies according to the providers. The values of QoS parameters exhibited here are for experiment purpose only and are not intended to reflect the real level of quality of these services. Machines used in our experiments are HP Compaq PC with Windows XP operating system. The configuration of each PC is 1.60 GHz Intel Pentium 4 processor and 2 GB RAM.

6.2.1 Service Monitor Ratings and Execution of Discovery Requests

For the experiment purpose, different QoS parameter values are provided for the same services on different timestamps so that service monitor can rate them based deviation between published QoS and monitored QoS. The monitor score of the service is calculated based on these ratings. Each consumer sends a discovery request to the discovery agent after each new rating is given and stored in ratings table. Hence, the monitor score of a service may be varying because a new monitor rating may be different from earlier one. In the starting of each experiment, there is only one monitor rating for each service. As the experiment progresses, a new rating is calculated and stored by the service monitor for each service on each timestamp. The number of ratings are equivalent to the number of timestamps for which monitoring was done.

6.2.1.1 Results obtained by specifying Only Functional Requirement

In the first experiment, for a customer C1, a service discovery request is executed for functional requirement only without specifying QoS and Monitoring of QoS requirement.

Table 6.1 Customer C1 Requirement	
Functional Requirement	Currency Converter Service
Max. no. of services	5

After specifying input as functional requirement only and maximum number of services for the selection, as a result service consumer C1 obtained 5 services for selection from functionally matched set of 50 web services WS1 to WS50 on the basis of maximum number of services to be returned and service publish timestamp. The ranking for these 5 services will be done on the basis of the time on which service was published in the UDDI registry. The recent one will be having top rank and so on.

Table 6.2 Services returned for Customer C1

Service Name	Response Time (seconds)	Reliability (%)	Availability (%)	Throughput (trans/sec)	Price ($)	Rank
WS50	0.05	98.00	98.00	300	3.00	1
WS49	0.06	99.00	99.00	400	5.00	2
WS48	0.07	99.90	98.00	400	4.00	3
WS47	0.03	99.00	99.90	200	6.00	4
WS46	0.04	99.99	98.50	300	6.00	5

Every time when a new service is published in the registry, it will match it for functional requirement and return the maximum number of latest services requested by the consumer. Consumer C1 requested 5 maximum numbers of services from the UDDI registry for the selection. He has to evaluate those 5 services on the basis of its non functional characteristics manually and select the appropriate service according to his requirement. Disadvantages of this type of service selection are as – (1) The services which are returned in the result for the selection are only based on their time of publishing, irrespective of their nonfunctional attributes. The chances of the best service selection are reduced as all the functionally matched services are not available for evaluation and selection. (2) As the evaluation of the services returned in the result is manual, it is tedious, time consuming and may be inaccurate. (3) The quality attributes of the services shown in the result are published by the service providers themselves at the time of service publishing/updating which may not trustworthy always and may vary over the time.

Hence, if Customer C1 sends discovery request for the same functionality 10 times with some days of time interval gap, during which many services matched with this functionality are published in the UDDI registry, he may get different results returning latest services irrespective of their appropriateness for him.

6.2.1.2 Results obtained by specifying Functional Requirement and QoS requirement

In this experiment, for a customer C2, a service discovery request is executed in which he specified a functional as well as QoS requirement without preference for QoS parameters as follows :

Table 6.3 Customer C2 Requirement	
Functional Requirement : Currency Converter	
QoS Requirement :	
QoS Parameter	QoS Value
Response Time (seconds)	0.05 second
Reliability (%)	99.0%
Availability (%)	99.0%
Throughput (trans/sec)	300 trans/sec
Price ($)	USD 0.02 per trans
Max. no. of services	5

When a Customer C2 sends a discovery request with the input as functional requirement, QoS requirement and maximum number of services for the selection, as specified in above table, he obtained the output with the result from functionally and QoS matched set of 50 web services WS1 to WS50 on the basis maximum number of services to be returned as shown in the following table. The ranking for these 5 services will be done on the basis of QoS Score calculated for each service. The service with the highest QoS score will be ranked first and so on.

The services along with QoS Score available for selection are as follows :

Table 6.4 Services returned for Customer C2

Service Name	Response Time (seconds)	Reliability (%)	Availability (%)	Throughput (trans/sec)	Price ($)	QoS Score
WS20	0.04	99.00	99.00	350	4.00	4.3610
WS23	0.05	99.50	99.90	400	3.00	4.6000
WS31	0.04	99.10	99.50	400	4.00	4.4920
WS40	0.03	99.10	99.50	300	4.00	4.4920
WS43	0.05	99.00	99.00	380	3.00	4.5910

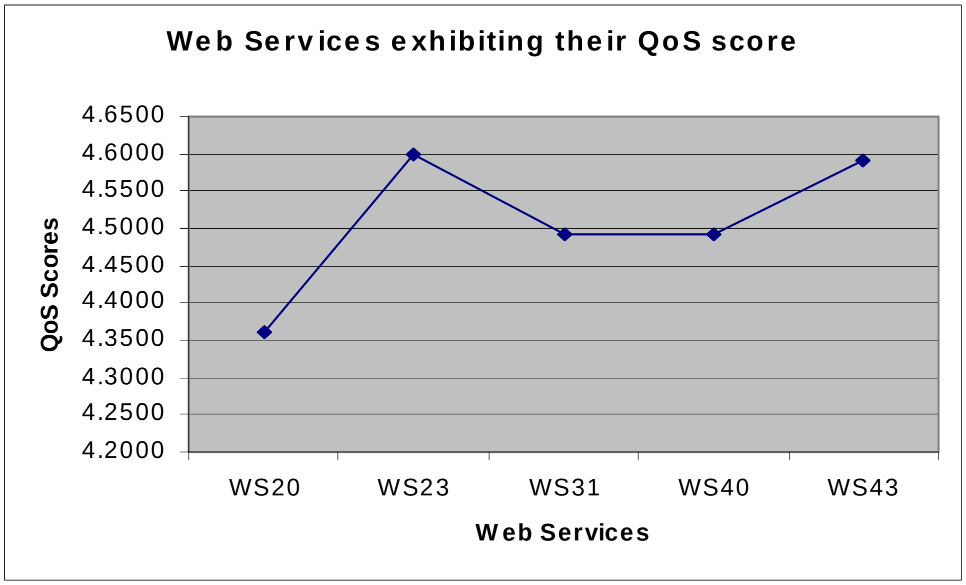


Figure 6.2 Graph of obtained web services with their QoS score for customer

As a result, the most appropriate web service with the highest QoS score is **WS23 (QoS score = 4.6000).**

In this experiment, from those services whose functional and QoS requirements are matched, are shortlisted for the final selection. For each of these service QoS score is calculated and based on this score set of services is sorted in ascending order. The maximum number of services to be returned in the result as specified in the consumer request, are shown to the service consumer in the output and then he will make the final service selection. Though, in this type of service selection, along with functional requirement, QoS of service is also taken into consideration for making the service available for the selection, some of the disadvantages in this type of service selection are as – (1) As QoS score is calculated giving equal weight to each QoS parameter, customer is unable to select the service according to his choice of parameters. E.g. in the above experiment QoS score of web service WS23 is highest (4.6000), whereas the response time of WS40 is highest(0.03 sec) as compared to WS23(0.05 sec) and rest of the QoS parameter values of WS40 are lesser than those of WS23. The probability of selecting web service WS23 is more than rest of the shortlisted services as its QoS score is higher. In this case, if the customer want a service with the preference order of QoS parameter as response time, availability, reliability, throughput and price, this selection won't allow him. (2) Another problem which remain as we have stated earlier that the quality attributes of the services shown in the result are published by the service providers themselves at the time of service publishing/updating which may not trustworthy always and may vary over the time.

6.2.1.3 Results obtained by specifying Functional Requirement and QoS requirement with QoS preference order

In this experiment, for a customer C3, a service discovery request is executed in which he specified a functional as well as QoS requirement with preference for QoS parameter as follows :

Table 6.5 Customer C3 Requirement

Functional Requirement : Currency Converter

QoS Requirement :

QoS Parameter	QoS Value	QoS Preference Order
Response Time (seconds)	0.05 second	1
Reliability (%)	99.0%	2
Availability (%)	99.0%	3

Throughput (trans/sec)	300 trans/sec	4
Price ($)	USD 0.02 per trans	5
Max. no. of services	5	

When a Customer C3 sends a discovery request with the input as functional requirement, QoS requirement along with the QoS parameter preference order and maximum number of services for the selection, as specified in above table, he obtained the output with the result from functionally and QoS matched set of 50 web services WS1 to WS50 as shown in the following table. The ranking for these 5 services will be done on the basis of QoS Score calculated for each service. In the calculation of QoS score, QoS parameter value as well as its preference order is also taken into consideration. The service with the highest QoS score will be ranked first and so on.

The services along with QoS Score available for selection are as follows :

Table 6.6 Services returned for Customer C3

Service Name	Response Time (seconds)	Reliability (%)	Availability (%)	Throughput (trans/sec)	Price ($)	QoS Score
WS20	0.04	99.00	99.00	350	4.00	12.9629
WS23	0.05	99.50	99.90	400	3.00	14.6000
WS31	0.04	99.10	99.50	400	4.00	13.4799
WS40	0.03	99.10	99.50	300	4.00	12.7299
WS43	0.05	99.00	99.00	380	3.00	14.3629

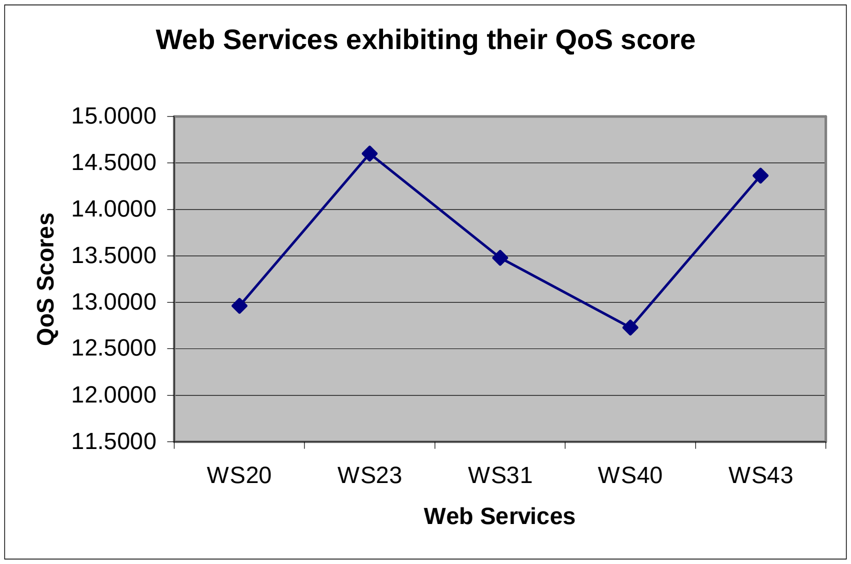


Figure 6.3 Graph of obtained web services with their QoS score for customer C3

As a result, the most appropriate web service with the highest QoS score is **WS23 (QoS score = 14.6000)** with the preference order as Price, Throughput, Availability, Reliability and Response Time.

The same experiment is executed for the customer C4 with the different preferences given for the QoS parameters having similar QoS values requirement given in below table :

Table 6.7 Customer C4 Requirement		
Functional Requirement : Currency Converter		
QoS Requirement :		
QoS Parameter	QoS Value	QoS Preference Order
Response Time (seconds)	0.05 second	5
Reliability (%)	99.0%	4
Availability (%)	99.0%	3
Throughput (trans/sec)	300 trans/sec	2
Price ($)	USD 0.02 per trans	1
Max. no. of services	5	

The services along with QoS Score available for selection are as follows :

Table 6.8 Services returned for Customer C4

Service Name	**Response Time (seconds)**	**Reliability (%)**	**Availability (%)**	**Throughput (trans/sec)**	**Price ($)**	**QoS Score**
WS20	0.04	99.00	99.00	350	4.00	13.2029
WS23	0.05	99.50	99.90	400	3.00	13.0000
WS31	0.04	99.10	99.50	400	4.00	13.4719
WS40	0.03	99.10	99.50	300	4.00	14.2219
WS43	0.05	99.00	99.00	380	3.00	12.8529

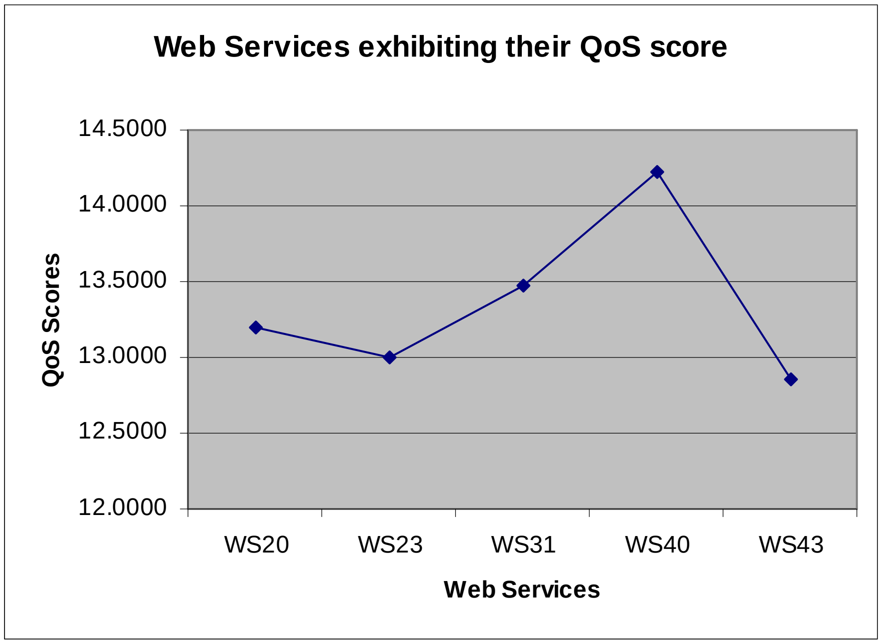


Figure 6.4 Graph of obtained web services with their QoS score or customer C4

As a result, the most appropriate web service with the highest QoS score is **WS40 (QoS score = 14.2219)** with the preference order as Response Time, Reliability, Availability, Throughput, and Price.

The same experiment is executed for the customer C5 with the different preferences given for the QoS parameters having similar QoS values requirement given in below table :

Table 6.9 Customer C5 Requirement		
Functional Requirement : Currency Converter		
QoS Requirement :		
QoS Parameter	QoS Value	QoS Preference Order
Response Time (seconds)	0.05 second	2
Reliability (%)	99.0%	4
Availability (%)	99.0%	3
Throughput (trans/sec)	300 trans/sec	5

Price ($)	USD 0.02 per trans	1
Max. no. of services	5	

The services along with QoS Score available for selection are as follows :

Table 6.10 Services returned for Customer C5

Service Name	Response Time (seconds)	Reliability (%)	Availability (%)	Throughput (trans/sec)	Price ($)	QoS Score
WS20	0.04	99.00	99.00	350	4.00	13.5779
WS23	0.05	99.50	99.90	400	3.00	14.2000
WS31	0.04	99.10	99.50	400	4.00	14.2219
WS40	0.03	99.10	99.50	300	4.00	13.4719
WS43	0.05	99.00	99.00	380	3.00	13.9029

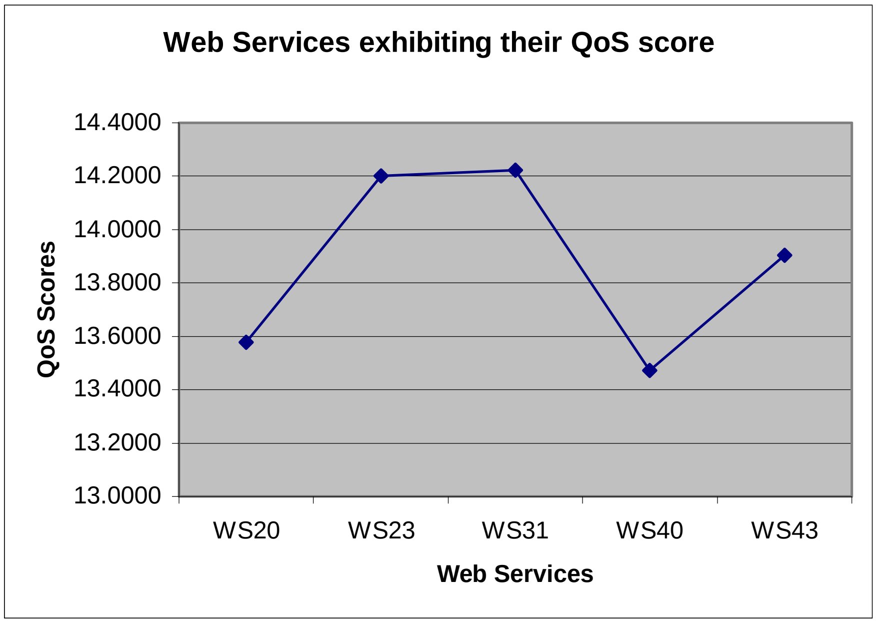


Figure 6.5 Graph of obtained web services with their QoS score for customer C5

As a result, the most appropriate web service with the highest QoS score is **WS31 (QoS** score = 14.2219) with the preference order as Price, Response Time, Availability, Reliability and Throughput.

In this experiment, according to the service consumer’s requirements, those services whose functional and QoS requirements are matched, are shortlisted for the final selection. For each of these services, QoS score is calculated considering QoS parameter value and its preference weight. A set of shortlisted services is sorted in descending order of this QoS score. From this sorted list of services, the maximum number of services to be returned in the result as specified in the consumer request, are shown to the service consumer in the output and then he will make the final service selection. Even though all the three customers C3, C4 and C5 in this experiment specify the similar functional and QoS requirements but different QoS parameter preference order, the most appropriate service for customer C3 is obtained as WS23, C4 is obtained as WS40 and C5 is obtained as WS31. From this it can be inferred that, QoS preference order has great impact on the service selection.

In this type of service selection, along with functional requirement, QoS of service and the QoS parameter preference order is also taken into consideration for making the service available for the selection. Hence the customers are allowed to prioritize their QoS requirement, but still the problem of trustworthiness of the QoS parameter values is still remaining and need to overcome.

6.2.1.4 Results obtained by specifying Functional requirement, QoS requirement and Monitor requirement

To overcome the problem of trustworthiness of the QoS parameter values published by the service provider themselves at the time of service publishing, we have introduced the concept of monitor who will be monitoring the published web services in the registry on regular time intervals by directly invoking the service through the service URL provided in web service description document which is stored in the UDDI registry. This monitor will rate the monitored services on those timestamps between 0 to 10 and these ratings are stored in the ratings table in the registry database. In this experiment, from the set of functionally and QoS matched services with the specified QoS parameter preference order, monitor ratings for the period of one month are considered for finding the most appropriate service for the consumer with the desired functionality and QoS parameter values. Again there is choice for the customer to give different weightages for QoS and Monitor ratings depending on whether he wants the service with highest QoS score value or with the assured QoS score value.

Table 6.11 Published QoS with QoS Score for each QoS matched service

Web Service Name	Response Time (seconds)	Reliability (%)	Availability (%)	Throughput (trans/sec)	Price ($)	QoS Score
WS20	0.04	99.00	99.00	350	4.00	13.2029
WS23	0.05	99.50	99.90	400	3.00	13.0000
WS31	0.04	99.10	99.50	400	4.00	13.4719
WS40	0.03	99.10	99.50	300	4.00	14.2219
WS43	0.05	99.00	99.00	380	3.00	12.9730

Ratings given to the shortlised services on monitoring timestamps are as shown in the following table :

Table 6.12 Actual QoS and Monitor Ratings for QoS matched services over one month

Web Service Name	Response Time (seconds)	Reliability (%)	Availability (%)	Throughput (trans/sec)	Price ($)	Monitor Rating	Timestamp
WS20	0.05	97.00	96.00	300	4.00	8	2013-06-03 10:10:42
WS20	0.06	96.00	97.00	320	4.00	7	2013-06-10 10:01:27
WS20	0.05	98.00	98.00	300	4.00	8	2013-06-17 10:12:15
WS20	0.04	98.50	96.00	340	4.00	10	2013-06-24 10:08:14
WS23	0.07	97.00	95.00	300	3.00	7	2013-06-03 10:25:33
WS23	0.08	96.00	96.00	320	3.00	6	2013-06-10 10:21:25
WS23	0.07	96.00	97.00	330	3.00	7	2013-06-17 10:28:22
WS23	0.08	95.00	96.00	310	3.00	6	2013-06-24 10:21:23
WS31	0.05	97.00	96.00	350	4.00	8	2013-06-03 10:46:29
WS31	0.06	96.00	95.00	370	4.00	7	2013-06-10 10:39:46
WS31	0.05	95.00	96.00	380	4.00	9	2013-06-17 10:46:16
WS31	0.06	97.00	97.00	360	4.00	7	2013-06-24 10:42:36
WS40	0.04	96.00	95.00	250	4.00	8	2013-06-03 11:04:23
WS40	0.05	95.00	97.00	200	4.00	5	2013-06-10 10:56:26
WS40	0.04	96.00	96.00	280	4.00	8	2013-06-17 11:05:03
WS40	0.05	97.00	97.00	220	4.00	6	2013-06-24 10:58:24
WS43	0.07	97.00	97.00	350	3.00	8	2013-06-03 11:22:38
WS43	0.08	95.00	95.00	280	3.00	6	2013-06-10 11:17:14
WS43	0.07	96.00	96.00	340	3.00	8	2013-06-17 11:19:17
WS43	0.06	98.00	96.00	320	3.00	9	2013-06-24 11:15:26

Table 6.13 Services returned with Overall score having different ranking for different weights given to QoS and Monitor Score

	Overall Score				
Web Services Name	**QoS weight = 0.5 & Monitor weight = 0.5**	**QoS weight = 0.3 & Monitor weight = 0.7**	**QoS weight = 0.7 & Monitor weight = 0.3**	**QoS weight = 0.1 & Monitor weight = 0.9**	**QoS weight = 0.9 & Monitor weight = 0.1**
WS20	0.964173754	0.978504252	0.949843256	0.992834751	0.935512757
WS23	0.850980755	0.825739968	0.876221542	0.800499181	0.901462329
WS31	0.943329199	0.941755095	0.944903303	0.940180991	0.946477406
WS40	0.909090909	0.872727273	0.945454545	0.836363636	0.981818182
WS43	0.925788141	0.931230460	0.920345821	0.936672780	0.914903502

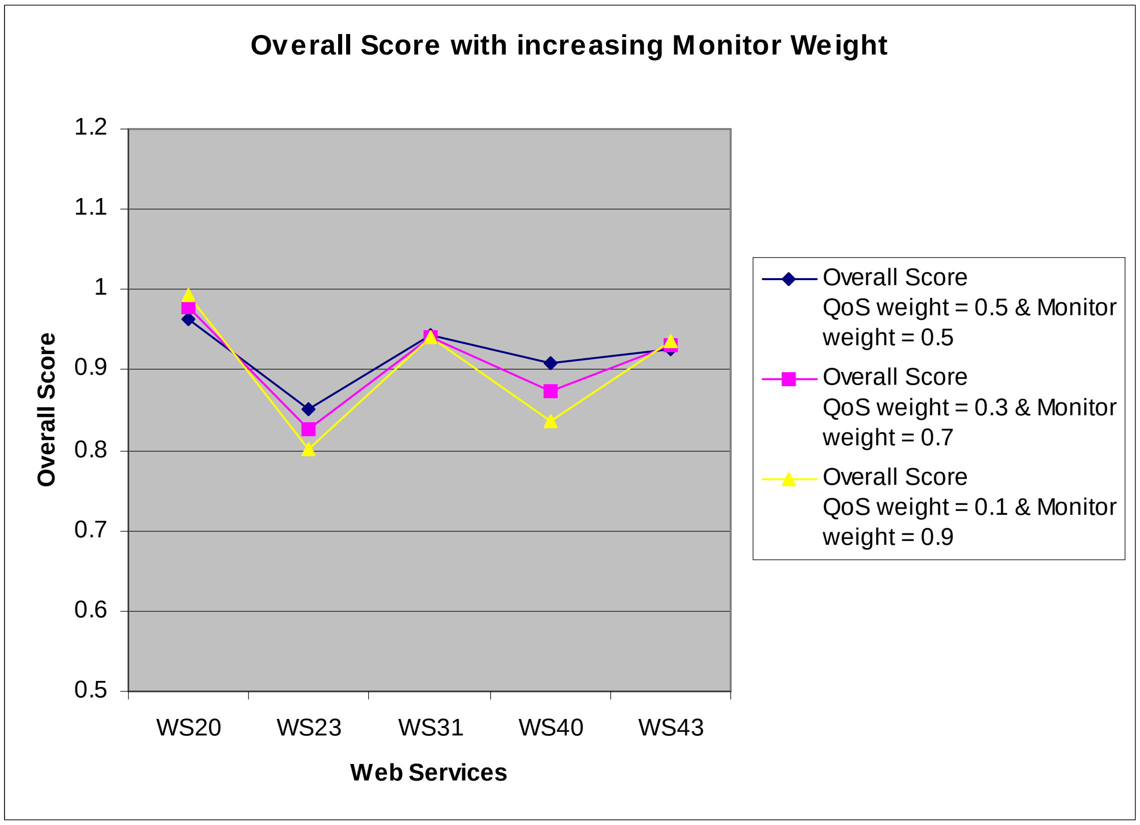


Figure 6.6 Graph of obtained web services with Overall score with increasing Monitor weight

In this case, web service WS20 is showing the best overall score if consumer gives equal or more weightage to Monitor ratings.

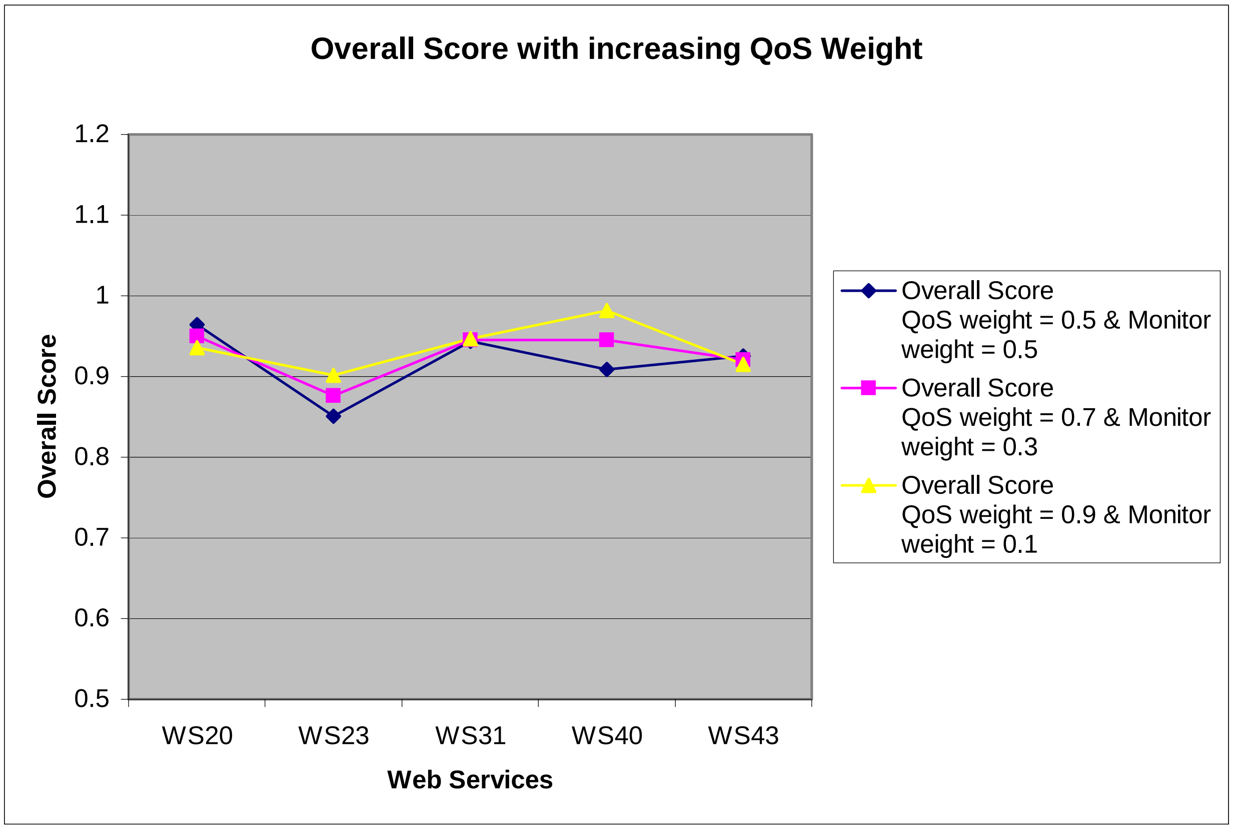


Figure 6.7 Graph of obtained web services with Overall score with increasing QoS weight

In this case, web service WS40 is showing the best overall score if consumer gives more weightage to QoS score obtained from published QoS parameter values.

Therefore, we can say that giving different weightage to Monitoring and published QoS parameter values the result may be affected as the candidate services for the selection are changed. Though it is obvious that more weightage given to monitor rating is appreciable, it is quite possible that ratings of the services may be changed over the time and its monitored QoS parameter values may be nearer to its published QoS parameter values for some time period and in that case given more weightage to monitor rating may miss out the best QoS web service from the selection.

6.2.1. 5 Effect of inclusion factor λ on monitor ratings

An inclusion factor λ is used to adjust the responsiveness of the monitoring score based on the changes in service behavior over the time period. We evaluate the effect of λ on the monitoring score and the result is shown in following figure. The monitoring score of a service is plotted as a function of the number of ratings provided by the monitor. When the inclusion factor is set to 0.25, only the recent ratings are taken into consideration for calculating the monitor score. Contrary to that when the factor is set to 0.75, more ratings are taken in the calculation of monitor score. The maximum inclusion factor value 1 makes the monitor score become stable but insensitive to changes in the ratings. The smaller inclusion factor vale 0 makes the monitor score respond quickly to changes in the ratings but may vary randomly.

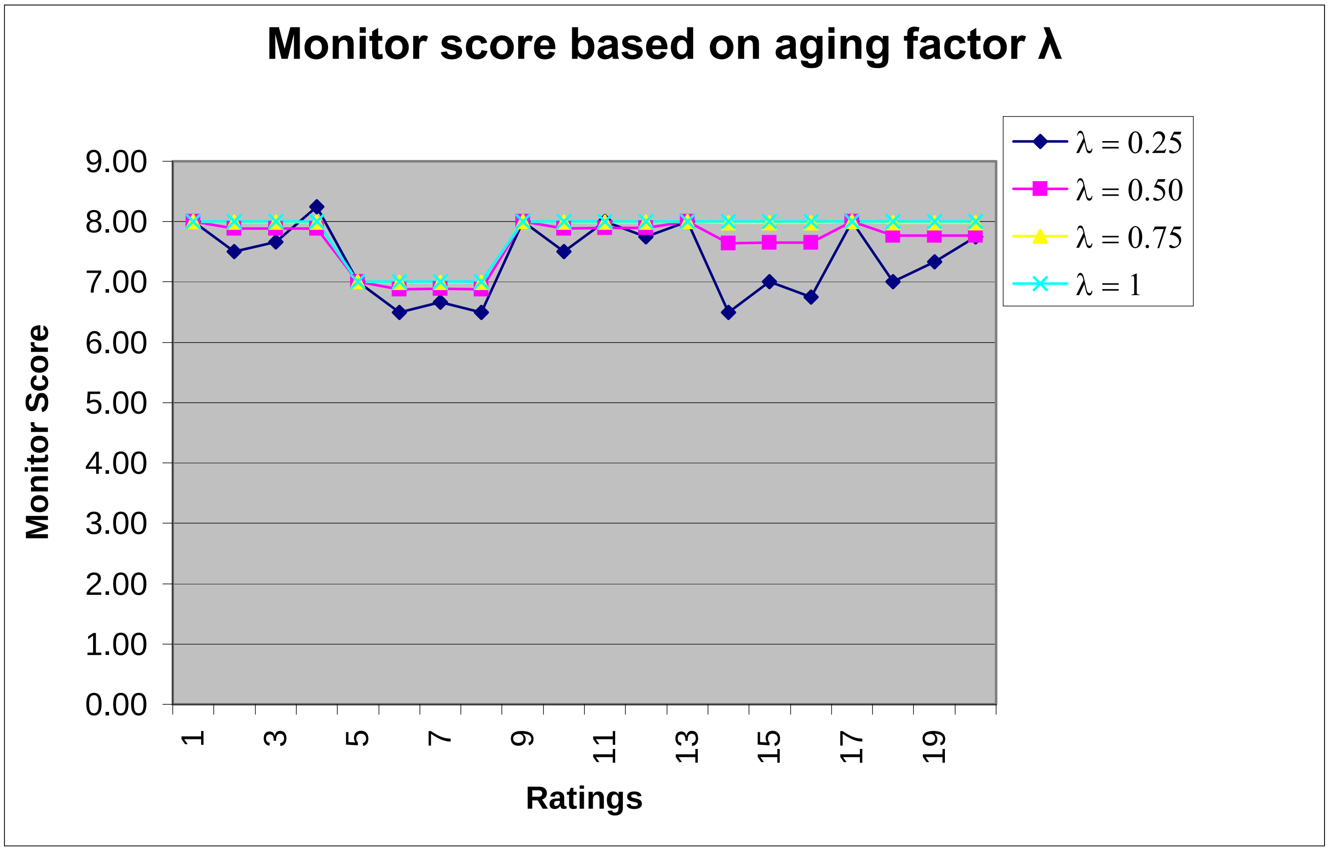


Figure 6.8 Graph showing effect of aging factor on Monitor score

6.3 Summary of Results Obtained

As can be seen from the above experiments' results and analysis,

1. Web service discovery by specifying functional as well as non-functional attributes ie. QoS parameter will yield better result as compared to only functional requirements.
2. Adding QoS requirement in the discovery request is not sufficient, but also customer should be able to specify the QoS preference order in the request.
3. By specifying the weightage of QoS score and monitored QoS score will narrow the search by matching them with the specified QoS score and monitor score weight.
4. Monitor ratings for a longer period will give more stable QoS which may be useful for the service selection.

Results are summarized more concretely in the next chapter.

Chapter 7

Summary and Conclusion

7.1 Summary

As a large number of web services with similar functionalities are published in the registry by different service providers nowadays, the service consumers are flooded with the services. Smart Web Service Discovery mechanism mainly emphasizes on finding the service that best fits the consumer’s requirement. For finding the best service that meets the consumer’s requirement, he must be able to specify his functional and QoS requirements along with his priorities. However, current UDDI registries do not provide any mechanism for service providers to publish and store the QoS information of their services in the registry. Moreover, the published QoS information of the services may not be always trustworthy and hence need some monitoring mechanism which will assure the service consumers about the QoS information published by the service providers. Also service consumers need a good registry browser tool which will be user friendly using which consumer will be able to specify his service discovery request with functional and QoS requirement with his priority for QoS for optimum service selection.

We are proposing a monitor-enhanced web service discovery model, Smart Web Service Discovery (SWSD). The QoS information published by the service provider is stored using *tModels* in a UDDI registry and is expressed in XML format. Using SWSD,when a service consumer sends a service discovery request, the discovery agent will find functionally and QoS matched services from the UDDI registry, retrieve monitor ratings, calculate monitor score and based on consumer’s requirement of monitor score, the optimum services will be ranked on the overall score (ie. QoS and Monitor score) and returned to the consumer. Service Monitor will be regularly monitoring the published services and rate them based on their actual QoS values on monitored timestamp.

7.2 Contribution

In this research study, we have examined some of the existing web service discovery mechanisms and identified some major issues and challenges in publishing service QoS information and finding the appropriate service that best meet the service consumer's requirement by matching published QoS with consumers QoS requirement. Also there was a major concern regarding trustworthiness of published QoS as it was published by the service providers themselves. As a solution, we have proposed a monitor-enhanced web service discovery model, Smart Web Service Discovery (SWSD). We have stated the service matching, rating, ranking and selection algorithm in this model to tackle the optimum service discovery issue.

We have developed a discovery model with a discovery agent and a service monitor. The discovery agent finds services those best satisfy a service consumer's functional, QoS and Monitor requirements that are specified in a discovery request. The Service Monitor monitors the published services by invoking them over the regular time interval and based on this, rates them on each monitored timestamp. These ratings are stored in the ratings table and fetched by the discovery agent to calculate a monitor score for each QoS and functionally matched services to rank them during the discovery process.

We have also developed a user-centric registry browser tool which will be assisting the service consumer to specify the functional as well as QoS and monitor requirements in a service discovery request which will also allow the consumer to specify the QoS priority.

To store published QoS information of services, a current feature in UDDI registry – *tModel* is used. When a business publishes a new web service, it creates a *tModel* in a UDDI registry. The QoS information of the web service is expressed in XML format in *tModel* which is referenced in a *bindingTemplate* of a web service.

We have stated service matching, rating, ranking and selection algorithms to find the services that match service consumer's requirements, to rate the services, to rank the matched services using their QoS and Monitor scores and to select services based on the service consumer's priority in the service discovery request.

After implementing the SWSD model, we have evaluated the model by conducting experiments. For the experiment purpose, before sending the service discovery requests, a large number of web services(approximately 1000) with different combinations – different functionalities and different QoS, similar functionalities and different QoS, similar functionalities and similar QOS, different functionalities and similar QoS were published in the UDDI registry. In the experiment, different consumers send the service discovery request with different functional, QoS and Monitor requirements. At the end we discussed the experimental results which demonstrate that the proposed service discovery model (SWSD) can find the most appropriate web service for the service consumer.

7.3 Conclusion

From the research study, we conclude that :

- The proposed Smart Web Service Discovery (SWSD) model provides a simple solution to a service discovery problem with less complexity at the same level of standards such as WSDL and UDDI as compared to other models based on WSLA.
- A Service Monitor in our model helps a service discovery agent to increase the chances of finding the services that provide assured QoS performance consistently and that match consumer's QoS and monitor requirements by assigning ratings to each service and providing those ratings to service discovery agent for optimum service selection.
- The monitor score based on historical service monitor ratings are playing crucial role in finding the service with high assurance of QoS and that best fit for the consumer's requirements. The sensitivity of monitor scores to changes in the monitor ratings is adjusted by the inclusion factor.

Summarized observation from the research work is given in the table below :

Table 7.1 : Summarized comparison of results obtained

Observation	**Existing UDDI registry browser - UB 0.2**	**SWSD with Functional match**	**SWSD with Functional and QoS match**	**SWSD with Functional, QoS match and QoS preference**	**SWSD with Functional, QoS match with QoS preference and Monitoring**
No. of web services found	50	05	05	05	05
Web Services	All functionally matched services WS1 – WS50	WS50, WS49, WS48, WS47, WS46	WS20, WS23, WS31, WS40, WS43	WS20, WS23, WS31, WS40, WS43	WS20, WS23, WS31, WS40, WS43
Service Ranking for selection	No service ranking. Services published recently are at the top of discovered service list	No service ranking. Services published recently are at the top of discovered service list	WS23 WS43 WS40 WS31 WS20 ∴Top ranked service : WS23	WS23 WS43 WS31 WS20 WS40 ∴Top ranked service : WS23	WS20 WS31 WS43 WS40 WS23 ∴Top ranked service : WS20
Relevancy of Services	Less	Less	More	More	More (with assured QoS)

7.4 Future Enhancements

There is a saying about software projects as - "*A software project is never finished, only abandoned*". Consequently, there is always scope for the improvements in the design and implementation of any software project. Some of the important issues that should be addressed in any future implementation or enhancement are listed below :

1. In future, integration of semantic information of services into Smart Web Service Discovery could be investigated in order to increase the flexibility and accuracy of the service discovery. Semantic-based service categorization and semantic-based

service selection will extend the service discovery from only syntactical information to a semantic level which may lead to more precision and relevance of the discovered services.

2. For automated service discovery, integrating Smart Web Service Discovery framework with the service consumer application would be one of the interesting project.
3. Smart Web Service Discovery may be further enhanced with the capability to allow the service consumer to specify their own QoS parameters and its values at the same time providing the default QoS parameters in absence of the user specified QoS parameters.
4. As service monitor regularly monitors all the published services in the UDDI registry and there may thousands of services registered in the registry in every month, there is large overhead on the monitor to rate each and every service though some of them may be continuously rated bad. Hence there is a need to improve Service Monitoring by keeping the services with consistent bad rating out of monitoring and increase the performance of discovery mechanism.

References

[1] Al-Masri, E.; Mahmoud, Qusay H., "Discovering Web Services in Search Engines," *Internet Computing, IEEE* , vol.12, no.3, pp.74,77, May-June 2008

[2] Al-Masri, E.; Mahmoud, Qusay H., "Interoperability among Service Registry Standards," *Internet Computing, IEEE* , vol.11, no.3, pp.74,77, May-June 2007

[3] Al-Masri, E.; Mahmoud, Qusay H., "Toward Quality-Driven Web Service Discovery," *IT Professional* , vol.10, no.3, pp.24,28, May-June 2008

[4] Al-Masri, E.; Mahmoud, Qusay H., "Web Service Discovery and Client Goals," *Computer* , vol.42, no.1, pp.104,107, Jan. 2009

[5] Al-Moayed, A.; Hollunder, B., "Quality of Service Attributes in Web Services," *Software Engineering Advances (ICSEA), 2010 Fifth International Conference on* , vol., no., pp.367,372, 22-27 Aug. 2010

[6] Anadiotis, G.; Kotoulas, S.; Lausen, H.; Siebes, R., "Massively Scalable Web Service Discovery," *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on* , vol., no., pp.394,402, 26-29 May 2009

[7] Apache Software Foundation. (2005). "Welcome to jUDDI". Retrieved May 23, 2006 from http://ws.apache.org/juddi/

[8] Artaiam, N.; Senivongse, T., "Enhancing Service-Side QoS Monitoring for Web Services," *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD '08. Ninth ACIS International Conference on* , vol., no., pp.765,770, 6-8 Aug. 2008

[9] Bisignano, M.; Di Modica, G.; Tomarchio, O., "JaxSON: A Semantic P2P Overlay Network for Web Service Discovery," *Services - I, 2009 World Conference on* , vol., no., pp.438,445, 6-10 July 2009

[10] Blake, M.B.; Sliva, A.L.; Muehlen, M.z.; Nickerson, J.V., "Binding Now or Binding Later: The Performance of UDDI Registries," *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on* , vol., no., pp.171c,171c, Jan. 2007

[11] Bonderud, P.; Sam Chung; Endicott-Popovsky, Barbara, "Toward Trustworthy Service Consumers and Producers," *Internet and Web Applications and Services, 2008. ICIW '08. Third International Conference on* , vol., no., pp.451,456, 8-13 June 2008

[12] Chakraborty, D.; Joshi, A.; Yesha, Y.; Finin, T., "Toward Distributed service discovery in pervasive computing environments," *Mobile Computing, IEEE Transactions on* , vol.5, no.2, pp.97,112, Feb. 2006

[13] Chenthati, D.; Mohanty, H.; Damodaram, A., "RDBMS for Service Repository and Matchmaking," *Intelligent Systems, Modelling and Simulation (ISMS), 2011 Second International Conference on* , vol., no., pp.300,305, 25-27 Jan. 2011

[14] Chi-Chun Lo; Ding-Yuan Cheng; Ping-Chi Lin; Kuo-Ming Chao, "A Study on Representation of QoS in UDDI for Web Services Composition," *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on* , vol., no., pp.423,428, 4-7 March 2008

[15] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGunneess, B. Barsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara [2004]: Bringing Semantics to Web Services: The OWL-S Approach. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition.

[16] D. Roman, H. Lausen, and U.Keller [2004]: Web Service Modelling Ontoloty - Standard. WSMO Working Draft, v0.2.

[17] D'Ambrogio, A., "A Model-driven WSDL Extension for Describing the QoS ofWeb Services," *Web Services, 2006. ICWS '06. International Conference on* , vol., no., pp.789,796, 18-22 Sept. 2006

[18] Diamantini, Claudia; Potena, Domenico; Cellini, J., "UDDI registry for Knowledge Discovery in Databases services," *Collaborative Technologies and Systems, 2007. CTS 2007. International Symposium on* , vol., no., pp.321,328, 25-25 May 2007

[19] Emekci, F.; Sahin, O.D.; Agrawal, D.; El Abbadi, A., "A peer-to-peer framework for Web service discovery with ranking," *Web Services, 2004. Proceedings. IEEE International Conference on* , vol., no., pp.192,199, 6-9 July 2004

[20] Eyhab Al-Masri and Qusay H. Mahmoud [2007]: Discovering the Best Web Service. WWW 2007.

[21] Eyhab Al-Masri, Qusay H. Mahmoud [2008]: Investigating Web Services on the World Wide Web. WWW 2008 Track: Web Engineering - Web Service Deployment.

[22] F. Banaei-Kashani, C.-C. Chen, and C. Shahabi. Wsdp [2004]: Web services peer-to-peer discovery service. International Conference on Internet Computing.

[23] Fuzhi Zhang; Yan Wang; Lin Wang, "A Web service discovery algorithm based on dynamic composition," *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on* , vol.2, no., pp.42,46, July 30 2007-Aug. 1 2007

[24] Gang Ye; Chanle Wu; Jun Yue; Shi Cheng, "A QoS-Aware Model for Web Services Discovery," *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on* , vol.3, no., pp.740,744, 7-8 March 2009

[25] Gang Zhou; Jianjun Yu; Rui Chen; Hui Zhang, "Scalable Web Service Discovery on P2P Overlay Network," *Services Computing, 2007. SCC 2007. IEEE International Conference on* , vol., no., pp.122,129, 9-13 July 2007

[26] Garcia, D.Z.G.; de Toledo, M.B.F., "A Web Service Architecture Providing QoS Management," *Web Congress, 2006. LA-Web '06. Fourth Latin American* , vol., no., pp.189,198, Oct. 2006

[27] Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A., "Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?", International Workshop on Web Engineering, 2004

[28] Giallonardo, E.; Zimeo, Eugenio, "More Semantics in QoS Matching," *Service-Oriented Computing and Applications, 2007. SOCA '07. IEEE International Conference on* , vol., no., pp.163,171, 19-20 June 2007

[29] Godse, M.; Bellur, U.; Sonar, R., "Automating QoS Based Service Selection," *Web Services (ICWS), 2010 IEEE International Conference on* , vol., no., pp.534,541, 5-10 July 2010

[30] Gorbenko, A.; Romanovsky, A.; Kharchenko, V., "How to Enhance UDDI with Dependability Capabilities," *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International* , vol., no., pp.1023,1028, July 28 2008-Aug. 1 2008

[31] Hicks, J.; Govindaraju, M.; Weiyi Meng, "Enhancing Discovery of Web Services through Optimized Algorithms," *Granular Computing, 2007. GRC 2007. IEEE International Conference on* , vol., no., pp.695,695, 2-4 Nov. 2007

[32] http://www.ebxml.org/specs/ebRS.pdf

[33] http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrim.pdf

[34] http://www.researchgate.net/publication/229010558_JRegistry_An_Extensible_UDDI_Registry

[35] http://www7b.software.ibm.com/dmdd/

[36] Hui Zhang; Weiying Gao, "A Research on QoS-based Ontology Model for Web Services Discovery," *Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on* , vol., no., pp.786,789, 23-25 Jan. 2009

[37] Huimin He; Haiyan Du; Dongxia Han; Yuemei He, "Research on the Models to Customize Private UDDI Registry Query Results," *Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on* , vol., no., pp.205,205, 18-20 June 2008

[38] Huiyuan Zheng; Jian Yang; Weiliang Zhao, "QoS Analysis and Service Selection for Composite Services," *Services Computing (SCC), 2010 IEEE International Conference on* , vol., no., pp.122,129, 5-10 July 2010

[39] Hunaity, M.A.R., "Towards an Efficient Quality Based Web Service Discovery Framework," *Services - Part I, 2008. IEEE Congress on* , vol., no., pp.261,264, 6-11 July 2008

[40] Ioan Toma, Brahmananda Sapkota, James Scicluna, Juan Miguel Gomez, Dumitru Roman, and Dieter Fensel (2005). "A P2P Discovery mechanism for Web Service Execution Environment". in Second WSMO Implementation Workshop

[41] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications* (SIGCOMM '01). ACM, New York, NY, USA, 149-160

[42] Jacek Kopeck´y [2007]: Semantic Web Service Offer Discovery. OTM 2007 Ws, Part I, LNCS 4805.

[43] Jianjun Yu; Rui Chen; Hao Su; Shengmin Guo, "Web Services Publishing and Discovery on Peer-to-Peer Overlay," *Services Computing, 2006. APSCC '06. IEEE Asia-Pacific Conference on* , vol., no., pp.329,334, Dec. 2006

[44] Jianxun Liu; Jie Liu; Lian Chao, "Design and Implementation of an Extended UDDI Registration Center for Web Service Graph," *Web Services, 2007. ICWS 2007. IEEE International Conference on* , vol., no., pp.1174,1175, 9-13 July 2007

[45] Jinghai Raoa, Dimitar Dimitrovb, Paul Hofmannb and Norman Sadeha [2006]: A Mixed Initiative Approach to Semantic Web Service Discovery and Composition :SAP's Guided Procedures Framework.

[46] Jiuxin Cao; Jingyu Huang; Guojin Wang; Jun Gu, "QoS and Preference Based Web Service Evaluation Approach," *Grid and Cooperative Computing, 2009. GCC '09. Eighth International Conference on* , vol., no., pp.420,426, 27-29 Aug. 2009

[47] Jorge Cardoso and Amit Sheth [2002] : Semantic e-Workflow Composition. Technical Report# 02-004, LSDIS Lab, Computer Science Department, University of Georgia, Athens GA, July 2002.

[48] K. Verma, K. Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, John Miller : METEOR–S WSDI [2003]: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services.

[50] Kawamura, T.; Hasegawa, T.; Ohsuga, A.; Paolucci, M.; Sycara, K., "Web services lookup: a matchmaker experiment," *IT Professional* , vol.7, no.2, pp.36,41, Mar-Apr 2005

[51] Keshan Zhu; Zhenhua Duan; Jianli Wang, "Quality of Service in Web Services Discovery," *Advanced Management of Information for Globalized Enterprises, 2008. AMIGE 2008. IEEE Symposium on* , vol., no., pp.1,5, 28-29 Sept. 2008

[52] Klein, M.; Bernstein, A., "Toward high-precision service retrieval," *Internet Computing, IEEE* , vol.8, no.1, pp.30,36, Jan-Feb 2004

[53] Lei Yang; Yu Dai; Bin Zhang, "Trustworthiness QoS Driven Service Selection in the Context of Environment," *Distributed Computing and Applications to Business Engineering and Science (DCABES), 2010 Ninth International Symposium on* , vol., no., pp.366,370, 10-12 Aug. 2010

[54] Libing Wu; Yanxiang He; Dan Wu; Jianqun Cui, "A Novel Interoperable Model of Distributed UDDI," *Networking, Architecture, and Storage, 2008. NAS '08. International Conference on* , vol., no., pp.153,154, 12-14 June 2008

[55] Magdalenic, I.; Pejakovic, I.; Skocir, Z.; Sokic, M.; Simunic, M., "Modeling ebXML registry service architecture," *Telecommunications, 2003. ConTEL 2003. Proceedings of the 7th International Conference on* , vol.2, no., pp.543,550 vol.2, 11-13 June 2003

[56] Mario Schlosser, Michael Sintek, Stefan Decker, Wolfgang Nejdl [2002]: A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services.

[57] Menasce, D., "QoS issues in Web services," *Internet Computing, IEEE* , vol.6, no.6, pp.72,75, Nov/Dec 2002

[58] Meng Li; Junfeng Zhao; Lijie Wang; Sibo Cai; Bing Xie, "CoWS: An Internet-Enriched and Quality-Aware Web Services Search Engine," *Web Services (ICWS), 2011 IEEE International Conference on* , vol., no., pp.419,427, 4-9 July 2011

[59] Mobedpour, D.; Chen Ding; Chi-Hung Chi, "A QoS Query Language for User-Centric Web Service Selection," *Services Computing (SCC), 2010 IEEE International Conference on* , vol., no., pp.273,280, 5-10 July 2010

[60] Ni Yulin; Si Huayou; Li Weiping; Chen Zhong, "PDUS: P2P-Based Distributed UDDI Service Discovery Approach," *Service Sciences (ICSS), 2010 International Conference on* , vol., no., pp.3,8, 13-14 May 2010

[61] Oracle9i Database Administrator's Guide

[62] Ouzzani, M.; Bouguettaya, A., "Efficient access to Web services," *Internet Computing, IEEE* , vol.8, no.2, pp.34,44, March-April 2004

[63] Overhage, S.(2002): On Specifying Web Services Using UDDI Improvements. 3rd Annual International Conference on Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World Net.ObjectDays, Germany

[64] Paliwal, A.V.; Shafiq, B.; Vaidya, J.; Hui Xiong; Adam, N., "Semantics-Based Automated Service Discovery," *Services Computing, IEEE Transactions on* , vol.5, no.2, pp.260,275, April-June 2012

[65] Pantazoglou, M.; Tsalgatidou, A.; Athanasopoulos, G.; Pilioura, T., "A Unified Approach for the Discovery of Web and Peer-to-Peer Services," *Web Services, 2006. ICWS '06. International Conference on* , vol., no., pp.901,902, 18-22 Sept. 2006

[66] Paolucci, M.; Sycara, K., "Autonomous Semantic Web services," *Internet Computing, IEEE* , vol.7, no.5, pp.34,41, Sept.-Oct. 2003

[67] Patrick C. K. Hung, Haifei Li [2003]: Web Services Discovery Based on the Trade-off between Quality and Cost of Service: A Token based Approach. ACM SIGecom Exchanges, Vol. 4, No. 2.

[68] Pei Li; Comerio, M.; Maurino, A.; De Paoli, F., "Advanced Non-functional Property Evaluation of Web Services," *Web Services, 2009. ECOWS '09. Seventh IEEE European Conference on* , vol., no., pp.27,36, 9-11 Nov. 2009

[69] Ping Wang; Kuo-Ming Chao; Chi-Chun Lo; Chun-Lung Huang; Yinsheng Li, "A Fuzzy Model for Selection of QoS-Aware Web Services," *e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on* , vol., no., pp.585,593, Oct. 2006

[70] Preeda Rajasekaran, John Miller, Kunal Verma, Amit Sheth [2004]: Enhancing Web Services Description and Discovery to Facilitate Composition.

[71] Qiang He; Jun Yan; Yun Yang; Kowalczyk, R.; Hai Jin, "A Decentralized Service Discovery Approach on Peer-to-Peer Networks," *Services Computing, IEEE Transactions on* , vol.6, no.1, pp.64,75, First Quarter 2013

[72] Qiang He; Jun Yan; Yun Yang; Kowalczyk, R.; Hai Jin, "Chord4S: A P2P-based Decentralised Service Discovery Approach," *Services Computing, 2008. SCC '08. IEEE International Conference on* , vol.1, no., pp.221,228, 7-11 July 2008

[73] Qianhui Liang; Chung, J. -Y, "A Federated UDDI System for Concurrent Access to Service Data," *e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on* , vol., no., pp.71,78, 22-24 Oct. 2008

[74] Richard Monson Haefel, The Ultimate Guide J2EE Web Services, Pearson Education, 2011

[75] Sapkota, B.; Roman, D.; Kruk, S.R.; Fensel, D., "Distributed Web Service Discovery Architecture," *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on* , vol., no., pp.136,136, 19-25 Feb. 2006

[76] Schulte, S.; Siebenhaar, M.; Steinmetz, R., "Integrating Semantic Web Services and Matchmaking into ebXML Registry," *Proceedings of the Fourth International Workshop SMR2 on Service Matchmaking and Resource Retrieval in the Semantic Web*; Shanghai, China, 2010

[77] Sellami, M.; Tata, S.; Maamar, Z.; Defude, B., "A Recommender System for Web Services Discovery in a Distributed Registry Environment," *Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on* , vol., no., pp.418,423, 24-28 May 2009

[78] Serhani, M.A.; Dssouli, R.; Hafid, A.; Sahraoui, H., "A QoS broker based architecture for efficient Web services selection," *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on* , vol., no., pp.113,120 vol.1, 11-15 July 2005

[79] ShaikhAli, A.; Rana, O.F.; Al-Ali, R.; Walker, D.W., "UDDIe: an extended registry for Web services," *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on* , vol., no., pp.85,89, 27-31 Jan. 2003

[80] Sharma, A.; Adarkar, H.; Sengupta, S., "Managing QoS through prioritization in Web services," *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on* , vol., no., pp.140,148, 13 Dec. 2003

[81] Shou-jian Yu, Xiao-kun Ge, Jing-zhou Zhang, Guo-wen Wu [2006]: Web Service Discovery in Large Distributed System Incorporating Semantic Annotations.

[82] Shuiguang Deng, Zhaohui Wu, Jian Wu and Ying Li [2004]: An Efficient Two-Phase Service Discovery Mechanism. WWW 2008, April 2008.

[83] Shuping Ran, "A Framework for discovering web services with Desired Quality of Services Attributes", IEEE International Conference on Web Services, Las Vegas, Nevada, USA, June 2003.

[84] Si Won Choi; Jin Sun Her; Soo Dong Kim, "Modeling QoS Attributes and Metrics for Evaluating Services in SOA Considering Consumers' Perspective as the First Class Requirement," *Asia-Pacific Service Computing Conference, The 2nd IEEE* , vol., no., pp.398,405, 11-14 Dec. 2007

[85] Sivashanmugam, K.; Verma, K.; Sheth, A., "Discovery of Web services in a federated registry environment," *Web Services, 2004. Proceedings. IEEE International Conference on* , vol., no., pp.270,278, 6-9 July 2004

[86] Skoutas, D.; Sacharidis, D.; Simitsis, A.; Sellis, T., "Ranking and Clustering Web Services Using Multicriteria Dominance Relationships," *Services Computing, IEEE Transactions on* , vol.3, no.3, pp.163,177, July-Sept. 2010

[87] Sycara, K.; Paolucci, M.; Soudry, J.; Naveen Srinivasan, "Dynamic discovery and coordination of agent-based semantic Web services," *Internet Computing, IEEE* , vol.8, no.3, pp.66,73, May-Jun 2004

[88] Tamilarasi, K.; Ramakrishnan, M., "Design of an intelligent search engine-based UDDI for web service discovery," *Recent Trends In Information Technology (ICRTIT), 2012 International Conference on* , vol., no., pp.520,525, 19-21 April 2012

[89] Thio, N.; Karunasekera, S., "Automatic measurement of a QoS metric for Web service recommendation," *Software Engineering Conference, 2005. Proceedings. 2005 Australian* , vol., no., pp.202,211, 29 March-1 April 2005

[90] Tian, M.; Gramm, A.; Naumowicz, T.; Ritter, H.; Freie, J.S., "A concept for QoS integration in Web services," *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on* , vol., no., pp.149,155, 13 Dec. 2003

[91] U.Keller, R. Lara, A. Polelres, I. Toma, M. Kifer, and D. Fensel [2004]: WSMO Web Service Discovery. WSMO Working Draft, v0.1.

[92] UDDI Spec TC, "UDDI Spec Technical Committee Draft, Dated 20041019" http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[93] UDDI Spec TC, "Using WSDL in a UDDI Registry, Version 2.0" - http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm

[94] Ulrich K, Birgitta K, Mirco S., Michael K. DIANE [2007]: An Integrated Approach to Automated Service Discovery, Matchmaking and Composition, International Conference on World Wide Web (WWW'07).

[95] Verma, K. Mulye, R. Zhong, Z. Sivashanmugam, K. and Sheth, A.: Speed-R: Semantic p2p environment for diverse web service registries. [Online] W3C Technical Report. Available from http://webster.cs.uga.edu/~mulye/SemEnt/Speed-R.html, 2004

[96] Vuong Xuan Tran; Tsuji, H., "QoS Based Ranking for Web Services: Fuzzy Approaches," *Next Generation Web Services Practices, 2008. NWESP '08. 4th International Conference on* , vol., no., pp.77,82, 20-22 Oct. 2008

[97] W3C Working Group (2004). "Web Services Architecture, W3C Working Group Note 11 February 2004". http://www.w3.org/TR/ws- arch

[98] W3C Working Group Notes (2004) "Web Service Architecture Requirements, W3C Working Group Note 11 February 2004". http://www.w3.org/TR/wsa-reqs

[99] Wenli Dong, "QoS Driven Service Discovery Method Based on Extended UDDI," *Natural Computation, 2007. ICNC 2007. Third International Conference on* , vol.5, no., pp.317,324, 24-27 Aug. 2007

[100] Ying Yin; Bin Zhang; Xizhe Zhang, "QoS-Driven Transactional Web Service Reselection for Reliable Execution," *Information Science and Management Engineering (ISME), 2010 International Conference of* , vol.2, no., pp.79,82, 7-8 Aug. 2010

[101] Youngkon Lee, "QoS Management for SOA by Synchronizing Quality Context in UDDI," *Future Generation Communication and Networking Symposia, 2008. FGCNS '08. Second International Conference on* , vol.1, no., pp.17,22, 13-15 Dec. 2008

[102] Youngkon Lee, "Quality Context Composition for Management of SOA Quality," *Semantic Computing and Applications, 2008. IWSCA '08. IEEE International Workshop on* , vol., no., pp.117,122, 10-11 July 2008

[103] Youngkon Lee, "Web Services Registry Implementation for Processing Quality of Service," *Advanced Language Processing and Web Information Technology, 2008. ALPIT '08. International Conference on* , vol., no., pp.538,543, 23-25 July 2008

[104] Yunsong Tan, "A Peer-to-Peer Based Web Service Discovery Mechanism," *Web Mining and Web-based Application, 2009. WMWA '09. Second Pacific-Asia Conference on* , vol., no., pp.175,177, 6-7 June 2009

[105] Yunsong Tan, "A Peer-to-Peer Based Web Service Discovery Mechanism," *Web Mining and Web-based Application, 2009. WMWA '09. Second Pacific-Asia Conference on* , vol., no., pp.175,177, 6-7 June 2009

[106] Zamanifar, K.; Zohali, A.; Nematbakhsh, N., "Matching Model for Semantic Web Services Discovery," *Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on* , vol., no., pp.50,54, 26-29 May 2009

[107] Ziqiang Xu; Martin, P.; Powley, W.; Zulkernine, F., "Reputation-Enhanced QoS-based Web Services Discovery," *Web Services, 2007. ICWS 2007. IEEE International Conference on* , vol., no., pp.249,256, 9-13 July 2007

Appendix – I

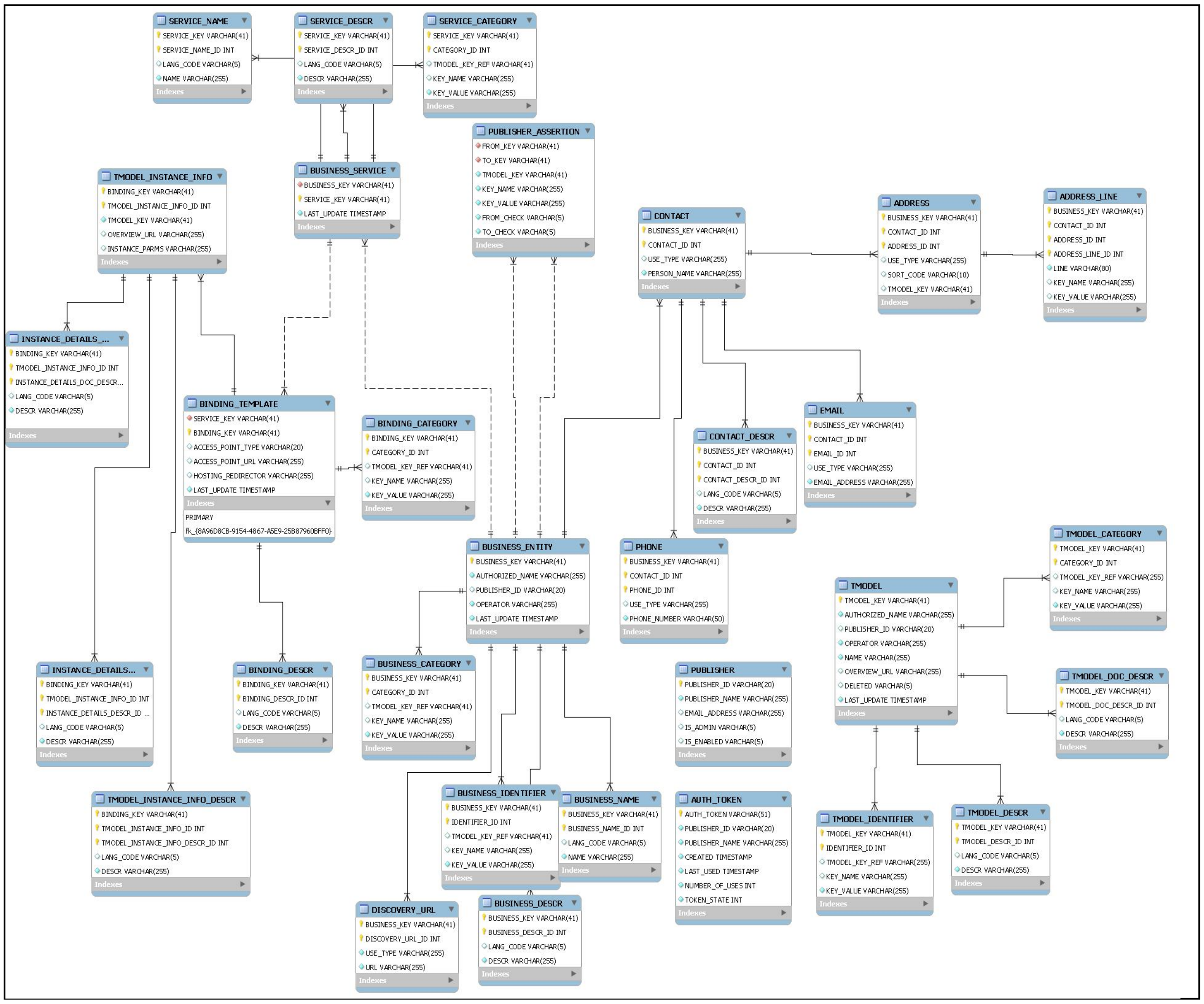
Glossary of relevant terms

- **Web Service :** A Web service is a reusable software component designed to support interoperable machine-to-machine interaction over a network.
- **RPC :** Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details..
- **SOA :** Service Oriented Architecture (SOA) is an application architecture in which all functions, or services, are defined using a description language and have invokable interfaces that are called to perform business processes.
- **DCOM :** Distributed Component Object Model (DCOM) is a proprietary Microsoft technology that allows Component Object Model (COM) software to communicate across a network.s
- **CORBA :** Common Object Request Broker Architecture (CORBA) is a specification developed by the Object Management Group (OMG) which describes a messaging mechanism by which objects distributed over a network can communicate with each other irrespective of the platform and language used to develop those objects
- **UDDI :** Universal Description Discovery and Integration (UDDI) is a set of specifications defining a registry service for Web services and for other electronic and non-electronic services. A UDDI registry service is a Web service managing information about service providers, service implementations and service metadata. Providers advertise their Web services on the UDDI registry. Consumers then use UDDI to discover Web services suiting their requirements and obtain the service metadata needed to consume those services.

- **WSDL :** Web Services Description Language (WSDL) is an XML-based language that describes Web services and their uses. It describes the abstract functionality of a service and provides a framework for describing the concrete details of a service description.
- **SOAP :** Simple Object Accsess Protocol (SOAP) is a protocol for implementing Web services. SOAP allow communication via the Internet between two programs, even if they run on different platforms, use different technologies and are written in different programming languages
- **QoS :** Quality of service which specify the non-functional properties of service.
- **ebXML :** Electronic business extensible markup language (ebXML) is an extensible markup language used to perform electronic business over the web. Enterprises conduct standard business by using ebXML over the Web through exchanging business messages, conducting trade relationships, communicating data in common terms and defining and registering business processes.
- **businessEntity :** A businessEntity entity contains descriptive information about a business or organization.
- **businessService :** A businessService contains descriptive information about a group of related technical services including the groupname, description and category information.
- **bindingTemplate** : A bindingTemplate contains information needed to invoke or bind to a specific service including the service URL, routing and load balancing facilities.
- **tModel** : A tModel is used to represent technical specifications such as service types, bindings and protocols. Also used to implement category systems that are used to categorize technical specifications and services.

Appendix – II

jUDDI database ERD



Appendix – III

Pilot Study Questionnaire

Questionnaire – 1

Questionnaire on the Quality of Web Services from Service Engineer's perspective

Based on your experiences as a service engineer who needs to find appropriate web services available over the internet, while designing and developing software applications, please provide information on how you perceive the quality of service you use in comparison to your expectations.

1. How many years have you been with the organization?
   - ☐ Less than a year
   - ☐ 1 - 3 years
   - ☐ 4 - 6 years
   - ☐ More than six years
2. What is the employee strength of your organization ?
   - ☐ Less than 100 employee
   - ☐ 100 - 500 employee
   - ☐ 501 - 1000 employee
   - ☐ More than 1000 employee
3. Your organization is providing services in which domain ?
   - ☐ Retailing
   - ☐ Tourists and Traveling
   - ☐ Healthcare
   - ☐ Insurance
   - ☐ Banking
   - ☐ Any other, please specify ___________________________________
4. Has your organization adopted web services ?
   - ☐ Yes
   - ☐ No

5. Adopting web service has reduced the cost of developing an application.
   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Agree
   - ☐ Strongly Agree
6. Applications are developed in-house and not outsourced from other company.
   - ☐ Yes
   - ☐ No
7. How many hits does your website record in a day?
   - ☐ Above 10000
   - ☐ Between 5001 - 10000
   - ☐ Between 1000 - 5000
   - ☐ Below 1000

8. How frequently customers complaint of slow response while performing transaction through your system?
   - ☐ Often
   - ☐ Sometimes
   - ☐ Rarely
   - ☐ Never
9. How frequently customers complaint of ‘Service temporarily unavailable’ issue through your system?
   - ☐ Often
   - ☐ Sometimes
   - ☐ Rarely
   - ☐ Never
10. How frequently customers complaint of ‘Transaction not completed successfully’ issue through your system?
    - ☐ Often
    - ☐ Sometimes
    - ☐ Rarely
    - ☐ Never

11. How frequently customers complaint of ‘System is too slow’ issue?
    - ☐ Often
    - ☐ Sometimes
    - ☐ Rarely
    - ☐ Never
12. An average cost of web service integrated in an application is acceptable as compared to developing the whole application.
    - ☐ Strongly Disagree
    - ☐ Disagree
    - ☐ Agree
    - ☐ Strongly Agree
13. Suggestion if any – ____

Name of Company:

Your Name:

Designation

Questionnaire – 2

Questionnaire on the Quality of Web Services from Service Consumer's perspective

Based on your experiences as an end user who uses online services available over the internet, please provide information on how you perceive the quality of service you use in comparison to your expectations.

1. For What purpose/purposes, you have used online services available over the internet from the following?
   - ☐ Shopping Books, CDs, Cloths, Footwear etc.
   - ☐ Railway Ticket Booking
   - ☐ Air Ticket Booking
   - ☐ Bus Ticket Booking
   - ☐ Internet Banking
   - ☐ Payment Gateway
2. How frequently you use online services available over the internet?
   - ☐ Daily
   - ☐ Weekly
   - ☐ Monthly
   - ☐ Rarely
   - ☐ Never
3. Are you happy with online services available over the internet ?
   - ☐ Yes
   - ☐ No
   - ☐ Can't Say
4. Which websites you prefer for using online services ?
   - ☐ www.amazon.in
   - ☐ www.easybillindia.com
   - ☐ www.makemytrip.com
   - ☐ Bank Portals
   - ☐ Any other, please specify ____________________________________

5. The response time of the most recent online services used by you was low.
   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Agree
   - ☐ Strongly Agree
6. While evaluating your most recent online service experience, the success rate of completing the transaction was high.
   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Agree
   - ☐ Strongly Agree
7. While evaluating your most recent online service experience, you find that online services were always readily available.
   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Agree
   - ☐ Strongly Agree
8. The charges incurred for using online services are nominal.
   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Agree
   - ☐ Strongly Agree
9. Rate the following parameters for online service selection on the scale of 1 to 5 (1 – Least significant, 5 – Most significant).

1.	Response Time	1	2	3	4	5
2.	Reliability	1	2	3	4	5
3.	Availability	1	2	3	4	5
4.	Throughput	1	2	3	4	5
5.	Price	1	2	3	4	5

10. Do you want to give equal weightage to all the parameters for service selection ?
    - ☐ Yes
    - ☐ No
    - ☐ Can’t Say
11. Suggestion if any –

    ____

Name :

Gender : ☐ Male ☐ Female

Age :

Qualification :

Profession :

Thank you for your feedback. I sincerely appreciate your honest opinion.

Appendix – IV

Research Paper Repository

- Published a paper in International Journal of Computer Science and Application, ISSN 0974-0767, Issue-III, December 2012 Edition on "Assessment of UDDI and ebXML Registry for e-Business Application".

- Published a paper in International Journal of Computer Applications, ISSN 0975-8887, January 2011 Edition on “Comparative Study of mechanisms for Web Service Discovery based on Centralized approach focusing on UDDI”.

- Published a paper in International Journal of Computer Science and Application, ISSN 0974-0767, Issue-I, January 2011 Edition on "Enhancing UDDI registry for storing Qos in tModel for discovering web services".

- Published a paper in “International Journal of Computer Science and Communication Volume-I, Number-II of September 2010”, ISSN 0973-7391 on “Ranking Web-services based on QoS for best-fit search”.

- Published a paper in “International Journal of Computer Science and Application”, ISSN 0974-0767, Issue-II, January 2010 on “Quantifying Web Services on Quality Parameters for Best-fit Web-service Selection”.

- Published a paper in International Conference IACC 2010 at Thapar University, Patiala on “Comparative Study of Centralized and Decentralized Approaches for Web Service Discovery Mechanism”.

- Published a paper in International Conference ICDM 2008 at IIM Ghaziabad, Delhi on “Model proposed for the senior management of an organization for utilizing resources effectively to adopt web services”.