A SYSTEM FOR ACCESS AND ADMINISTRATION OF FAULT-TOLERANT WEB SERVICES

A Thesis

Presented to the Faculty of the Graduate School

University of Missouri-Columbia

In Partial Fulfillment of the Requirements for the Degree

Master of Science

---

by

ABDULKADAR A. KHAMBATI

Dr. Gordon K. Springer, Thesis Supervisor

JULY 2005

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled

A SYSTEM FOR ACCESS AND ADMINISTRATION OF FAULT-TOLERANT WEB SERVICES

Presented by

Abdulkadar A. Khambati

A candidate for the degree of

Master of Science

And hereby certify that in their opinion it is worthy of acceptance.

Gordon K Springer

Dr. Gordon K. Springer

[illegible]

Dr. Wenjun Zeng

[illegible]

Dr. Timothy B. Patrick

Acknowledgements

This has been a long journey which would not have consummated without the assistance of many people. I would like to express sincere gratitude and appreciation to them who were supportive in different ways.

First and foremost, I would like to thank my thesis advisor, Dr. Gordon Springer, for his immense patience, unfaltering guidance, constructive criticism, deep insight, and valuable time without which I would have been certainly lost on the path. His vast knowledge and experience helped me over the many blockages in the culmination of my thesis.

I appreciate the valuable time of Dr. Wenjun Zeng and Dr. Timothy Patrick for serving on the thesis committee. I would like to thank my friend, Jim Ries, who lent me his book that sparked my interest in Web Services.

I am forever indebted to my parents, Abbas and Kulsum Khambati, for their unconditional support, love, and constant encouragement that helped me not only to be successful, but also to excel. I am also grateful to my wife, Nisreen, for her support, endless patience, and understanding. Lastly, I would like to thank my brothers, Hakim and Huzefa, for their support.

A SYSTEM FOR ACCESS AND ADMINISTRATION OF FAULT-TOLERANT WEB SERVICES

Abdulkadar A. Khambati

Dr. Gordon K. Springer, Thesis Supervisor

ABSTRACT

Web Services, a recent development in web technology, enables programs (services) on the World Wide Web (WWW) to communicate with each other. A program (web service) available on the WWW has to be robust to serve the vast number of users on the WWW and should have a high level of availability. This can be achieved by incorporating fault-tolerance in a program by having multiple instances of the same program running on different servers. This thesis presents a system to start, administer and monitor multiple instances of a web service on different servers. The system designed to incorporate fault-tolerance in web services cannot afford to have a single point of failure. Hence, the system itself is designed to be fault-tolerant. This thesis also implements a Web Service Registry that provides a means for the developers to register their web services with the Web Service Registry and thus, make the web services accessible to the world.

Table of Contents

List of Figures

List of Tables

1. Introduction

A vast amount of information is available on the World Wide Web (WWW). This information is easily accessible in the form of static and dynamic web pages that can be retrieved using a web browser. In the WWW terminology, the entity (machine) that is being requested for information or data is called a server. The entity (machine) requesting information or data is called a client. A static web page consists of static data that is formatted using Hyper Text Markup Language (HTML), a language used by web browsers to interpret and present data as a web page. A static web page is generated beforehand and is constant in its existence. More information on the conventions followed by HTML to format data to be presented as a web page can be found in "Special Edition Using HTML" [2].

With the introduction of the Common Gateway Interface (CGI) [5], it became possible to execute a program on the server, in real time, upon receiving a request from the client. The program generates output, based upon the request and provides the client with a web page containing dynamic content. The dynamic content (data) is generated based upon database queries, various computations and other information retrieval procedures specific to the request rather than the constant content provided by static web pages.

Entering a Universal Resource Locater (URL) string in a web browser accesses these static and dynamic, web pages. A URL is a unique name given to an object accessible on the WWW. The object can be a static web page, a program that generates a dynamic page or any other entity accessible on the WWW.

The CGI led to the evolution of web applications that can be generically defined as programs residing on remote machines. These can be called by entering the URL string in a web browser or by submitting a HTML form that has the URL of the program embedded in it. HTML forms are simple data entry forms that are displayed in a web browser and are used to gather information from the client to be sent to the remote programs (web applications). Thus, web browser is a primary means to access static web pages or programs that generate dynamic web pages on the WWW.

With the advent of a recent technology called Web Services (WS) [1], the same information on the WWW can be made available programmatically. WS provides clients the ability to interact with remote programs on the WWW from within their own programs. The communication is possible among the programs regardless of the programming language they are built with and the operating system they are implemented on. Since WWW can be thought of as a collection of data and programs running on different machines, WS facilitates interoperability among the heterogeneous programs (web applications) available on the WWW.

In the previous paragraph the words “Web Services” referred to the technology. Programs that are built using this technology are also termed as “web services” (services available on the web). Henceforth, “web service” refers to a program that is built using the technology.

Web services use Service Oriented Access Protocol (SOAP), a protocol represented using a text-based language, eXtensible Markup Language (XML) [7], to communicate with each other. It can be sent over any transport protocol such as HTTP [4], TCP [8], UDP [8], or other communication protocol. A more detailed explanation of SOAP and HTTP is provided in Chapter 2.

The process of calling another program (service) on a remote machine is called a Remote Procedure Call (RPC). There are various protocols in the industry that support RPC. The Distributed Computing Environment (DCE) [3] implements the DCE-RPC Network protocol to support RPC on various platforms such as UNIX, Windows and other platforms in a DCE environment. Microsoft has its Distributed Component Object Model (DCOM) [19], a high-level protocol based on the DCE-RPC protocol. These are binary (non-text) protocols and each has its own data representation. The data communicated between programs is in a binary format and hence, the client and the server must both use the same binary protocol to interpret the data correctly. The data, if needed, is translated to a common data representation understood by both the client and the server platforms. This translation is done using a Network Data Representation (NDR) [3] and other prototypes. SOAP on the other hand is a text-based protocol that defines a new standard of transferring data from one point to another. The data exchanged between programs using SOAP is represented in XML, a text format, rather than the binary format used by the aforementioned protocols. SOAP can be sent over HTTP. Since HTTP is the most commonly used transport protocol on the WWW, SOAP facilitates building of programs (web services) on the WWW that can be accessed by other programs that understand XML and support HTTP.

Any program that understands text and supports HTTP can be a client of a web service. Thus, a web service can be expected to have a large number of clients such as the WWW. Any program (web service) that has a huge number of clients should be able to serve the vast number of clients and should be able to function properly in the event of failures. This can be achieved by making a program fault-tolerant. Fault-tolerance can be defined as the ability of a program to serve the users in the event of failures. It can be accomplished by having multiple instances (replicas) of the same service running on different machines such that even if one instance fails there are one or more instances available to serve the requests. Having replicas would not only protect the service from a single point of failure, but would also assist in serving requests simultaneously by distributing the requests among the multiple instances. Thus, by having multiple instances of the same service running on different servers (machines) and having at-least one copy of the service available, a web service can be made fault-tolerant.

The goal of this thesis is to design and build a system to administer and monitor multiple instances of a web service and thus, incorporate fault tolerance in web services. Due to multiple instances of the same web service running on different servers (machines), a central Web Service Registry (WSR) is needed to store the access points of the multiple instances of the web service. An access point of a web service can be defined as a unique name like a URL to access an instance of the web service. The central WSR is contacted by clients to look up the access points of the instances of a web service. The WSR is updated with the access points in case a new instance is started and/or a running instance of the web service fails. A WSR also serves as a medium for web service providers to advertise their web services to the clients. This thesis also focuses on building a fault-tolerant WSR and providing access to the fault-tolerant web services.

The system to start and administer multiple instances of a web service, along with the Web Service Registry, provides a platform to not only incorporate fault tolerance in web services but also advertise them available to the world. This is demonstrated in the thesis by starting and administering a web service using the system and making it available to clients by registering it with the WSR. A client application is also built to access the fault-tolerant web service.

The system to start and administer fault-tolerant web services must be fault-tolerant itself. Thus, it is required that the components used to build the system have multiple instances running on different machines. Primarily there are two basic components: a manager that administers and monitors the web service instances running on different machines and, a database that contains the information used by the manager and the Web Service Registry.

The manager is further comprised of a Web Service Manager (WSM) and a fault manager. The Web Service Manager is responsible for starting and administering the instances of a web service. The WSM is built as a Distributed Computing Environment (DCE) server. The Distributed Computing Environment (DCE) provides an environment for programs (applications) to interact in a distributed environment. DCE makes it possible to incorporate fault tolerance in DCE servers by allowing replicated DCE server on multiple machines and having a Cell Directory Service (CDS) to look up the hostnames of the machines that run the DCE servers. The CDS stores the names of the machines running a DCE server and is contacted by clients to locate the machine running a particular DCE server. Thus, the Web Service Manager is run on multiple machines as a DCE server to provide fault-tolerance to the system. A detailed explanation of the DCE client-server interaction and DCE CDS is provided in Chapter 2.

The Web Service Manager is responsible for starting multiple instances of a given web service, starting a new instance of the service on a back-up or standby server (machine) whenever a running instance fails. A standby server (machine) is a server that is instructed to start a web service instance in the event of a failure of an instance on any other server (machine). Whenever an instance fails, the failed instance tries to notify the Web Service Manager of its failure. Upon receiving a notification of failure, the Web Service Manager starts another instance on a back-up server (machine) to fill in for the failed instance. The list of standby servers available to a web service is read from the database by the Web Service Manager. Thus, at any given time, the Web Service Manager tries to keep the same number of instances available to serve client requests.

The other component of the manager, besides the WSM, is the fault manager. The fault manager is a program (Monitor) that periodically checks the availability of the multiple instances of a web service started by the Web Service Manager. If an instance is no longer running, the Monitor tries to start another instance on a standby or back-up server (machine). Thus, the failure of a web service instance is detected by the Monitor in case the instance is not able to notify the Web Service Manager. The Web Service Manager and the Monitor both update the access points of the web service in the database as soon as an instance fails or a new instance is started. Thus, clients using the WSR get updated access points of the fault-tolerant web services.

The other basic component of the system besides the manager is the database. Fault tolerance at the database level in the system is achieved by having multiple copies of the database running on different servers (machines). All the copies of the database are communicated with simultaneously using multicast [9], a communication method in which a message is transmitted to selected receivers on the network listening on a given multicast address. A detailed explanation of multicast is given in Chapter 2. An update (write) operation is simultaneously transmitted to all the copies of the database using multicast. A read operation is performed by remotely connecting to one of the copies of the database. A database that can be remotely connected was required for this thesis. Thus, the Oracle database was chosen. The technicalities involved in communicating with the Oracle database to do a read and a write operation is explained later in the thesis.

Within the system, multicast is used to communicate between the database instances. The other components within the system, however, interact with each other using DCE-RPC, and HTTP and SOAP protocols. Thus, for the fault-tolerant system to be operable, only the database instances are required to be deployed on a multicast-enabled network. Multicast works on a Local Area Network (LAN), but the messages are not transmitted to other network segments if the routers between the segments are not multicast-enabled.

The chapters in this thesis have been arranged to provide a systematic presentation of the design and the usage of the fault-tolerant system and, easy understanding of the technologies and concepts used in the implementation of the system. The outline of the remainder of this thesis is as follows.

Chapter 2 includes a detailed explanation of the protocols and technologies. Chapter 3 discusses the design of the system where the components of the system are introduced and their roles discussed. Chapter 4 demonstrates the usage of the system to start and administer fault-tolerant web services along with the performance evaluation of the fault-tolerant system to administer fault-tolerant web services. Chapter 5 discusses the Web Service Registry and exemplifies the usage of the Web Service Registry. Chapter 6 gives conclusions derived from this thesis and discusses future work.

2. Protocols and Technologies

In order to understand the design and the implementation of the fault-tolerant system, it is necessary to understand the different protocols and technologies used by the system. This chapter describes the protocols that were briefly mentioned in Chapter 1 and provides the background needed to understand the design and the implementation of the system. The discussion begins with the DCE-RPC Network protocol and then moves on to discuss multicast. The chapter concludes with a detailed explanation of Service Oriented Access Protocol (SOAP).

2.1 DCE-RPC

The Distributed Computing Environment (DCE) provides an environment for programs (applications) to interact in a distributed environment. The DCE facilitates the remote client-server interaction where the client and the DCE server are running on different machines. A DCE server implements a set of procedures that can be remotely called by clients. One of the important components of DCE is the DCE Cell Directory Service (CDS), which maintains information about the DCE servers running in the distributed environment. The CDS stores the hostnames of the machines where the DCE servers are running. The DCE servers can register themselves in the namespace maintained in the CDS, so that the client can look up the hostname of the machine running a DCE server by contacting the CDS. Thus, by replicating the DCE servers on different machines and registering the DCE servers in the namespace maintained in the CDS, a single point of failure of the DCE server is avoided.

DCE software provides an integrated set of tools and services to build applications in the DCE environment. It provides a high level interface to the lower-level aspects of communication that needs to be dealt with when two programs communicate in the DCE environment; such as converting the data to the format understood by heterogeneous host machines.

The following sections discuss the Remote Procedure Call (RPC) method for a client and a DCE server to communicate, the initialization steps followed by a DCE server to register itself with the CDS and, the steps taken by a client to look up and communicate with a DCE server.

2.1.1 Remote Procedure Call

A Remote Procedure Call (RPC) is a concept of calling a procedure on a remote machine in a similar manner to a local procedure. The intricacies involved in making a remote procedure call to the remote machine (DCE server) are hidden from the application. The DCE software provides RPC development tools that automatically generate special code, called stub code, which manages the communication between the calling program and the DCE server. The stub code, both on the client and the server, and RPC runtime libraries convert data to the appropriate formats and perform the communication between the calling program and the DCE server. A DCE server has an Interface Definition file that contains the definition of the remote procedures implemented by the DCE server. The stub code for the client and the server is generated by compiling the Interface Definition file by an Interface Definition Language (IDL) compiler. The Interface Definition file contains a Universal Unique Identifier (UUID), which is unique for a given DCE server. The term UUID is used in many places in this thesis. The UUID is a value which is guaranteed to be universally unique. It is used to identify an entity, which can be a machine, a program or any object of significance.

Figure 2.1 shows the execution of a RPC. The main program calls a procedure as if it was a part of the same program [step 1]. The call is transferred to the client's stub. The client's stub accepts the input arguments to the procedure and marshals them into a format understood by the server's stub [step 2]. Marshalling of data means converting data from the local byte-representation format into the network byte-representation format that is understood by different platforms across the network. The client's stub uses routines from the RPC runtime library to find and locate the DCE server on the network. If the client has the address of the host machine running the DCE server, the client directly communicates with that machine or, it contacts the CDS to get the host machine's address.

After the DCE server is located, RPC runtime library is called to transmit the data over the network to the machine where the DCE server is running [step 3]. The RPC runtime library on the server listening for procedure calls from the clients dispatches the call to the proper procedure in the DCE server's stub [step 4]. The server's stub unmarshals the data to a format understood by the machine and calls the required procedure [step 5]. Unmarshalling of data means converting data from the network byte-representation format to the local byte-representation format understood by the local system. The procedure executes and passes the output to the server's stub [step 6]. The server's stub prepares the output by marshalling the data [step 7] and calls the RPC runtime library to transmit the data back to the client's stub [step 8]. The RPC runtime library on receiving the data from the DCE server passes it to the client's stub [step 9]. The client stub code unmarshalls the data and sends it to the calling program [step 10]. The calling program gets the result sent by the remote procedure [step 11]. Thus, RPC development and execution tools provide an abstraction layer for the application to make remote procedure calls without worrying about the details of the communication over the network.

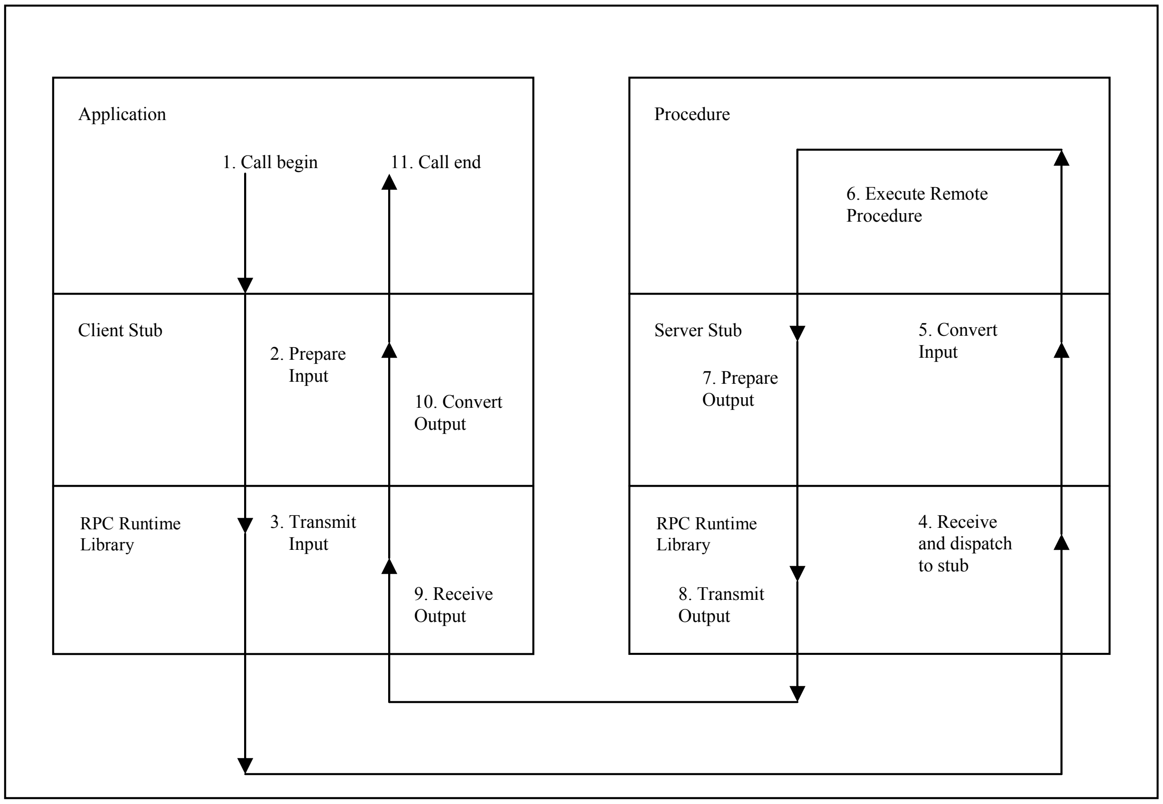


Figure 2.1: Execution of a Remote Procedure Call

A DCE server has to identify (register) itself with the RPC runtime library on the local host machine where the DCE server is running. This is done for the RPC runtime library on the local machine to accept remote procedure calls from the clients and forward them to the DCE server. The following sub-section discusses the initialization steps followed by a DCE server to make itself known to the RPC runtime library and to the clients.

2.1.2 DCE Server Initialization

A DCE server, during its initialization process, registers its Interface with the host machine's RPC runtime library. A DCE server has an Interface Definition file that contains the definition of the remote procedures implemented by the DCE server. Interface registration instructs the RPC runtime library on the host machine to expect calls to the Interface being registered. After Interface registration, the DCE server registers the communication or transport protocols the host machine will support for the remote procedure calls such as TCP, UDP and others. Protocol registration also causes the DCE server to create communication endpoints that the clients use to communicate with the DCE server. An endpoint is a process address (such as a port number) of a DCE server on the host machine. After a DCE server has registered itself with the runtime library, it should make the binding information available to the clients. The binding information is the information a client needs to find a DCE server. One of the ways of making the binding information available to the clients is by registering the DCE server in the namespace of the CDS.

As shown in Fig 2.2, the DCE server stores the binding information that includes the name of its Interface and the host machine's address in the CDS [step 1]. The local host machine (where the DCE server is running) maintains a list of the DCE servers running on the local machine and their respective endpoints in a database called the host endpoint map. A RPC daemon process manages the endpoint map and resides on a well-known endpoint (port 135) to listen for requests from clients. The initializing DCE server writes its endpoint and the Interface in the local host endpoint map [step 2]. After writing into the CDS and local host endpoint map, the DCE server listens on the port associated with the endpoint [step 3].

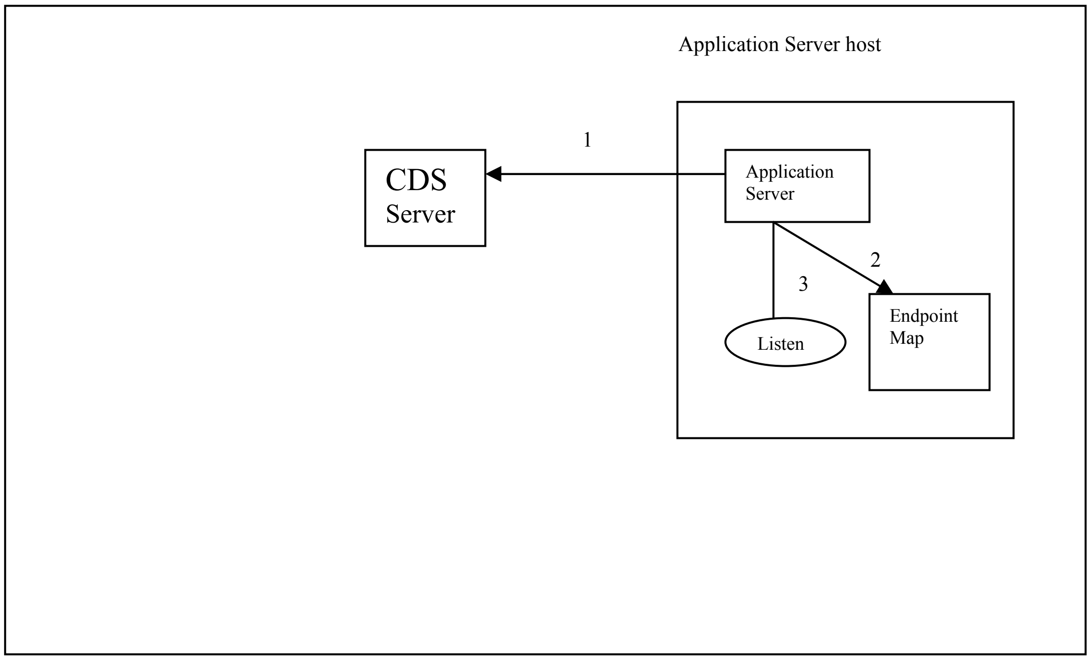


Figure 2.2: DCE Server Initialization

2.1.3 Locating and Communicating with a DCE Server

Figure 2.3 shows the step-by-step process of finding and communicating with a DCE server. The RPC library on the client interacts with the CDS software on the client to get the hostname of the machine running a DCE server implementing a particular Interface. The CDS software on the client contacts the CDS to get the binding information of a DCE server implementing that Interface. The CDS server returns the address of the host machine to the CDS software on the client if it finds a match for the Interface in its namespace [step 1]. After finding the hostname of the DCE server, the client contacts the RPC daemon running on the host machine [step 2]. The RPC daemon runs on a well-known port (135) so the client can communicate with it. If an endpoint is found for that Interface, the RPC daemon forwards the request to the designated DCE server [step 3]. Upon receiving the request from the RPC daemon, the DCE server responds to the client [step 4]. As part of the response, the DCE server sends its endpoint information to the client. Once the endpoint information is available to the client, the client directly corresponds with the DCE server using the endpoint information [Step 5]. More information about the server initialization and server look up is available in “Understanding DCE” [3].

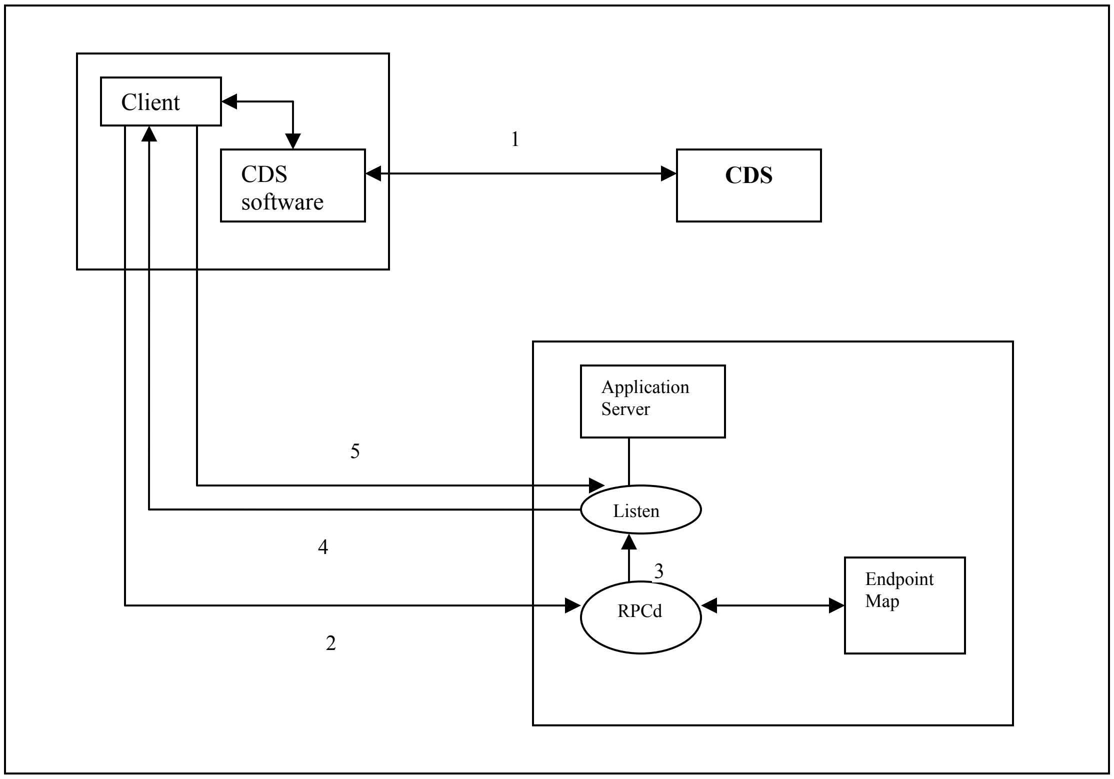


Figure 2.3: Locating and Communicating with a DCE Server

2.2 Multicast

The Internet Protocol (IP) [8], a protocol used for the transmission of data over a network, supports three types of transmission or communication methods, unicast, broadcast and multicast. IP uses a different class of address to accommodate each of the three transmission methods. This section briefly talks about the two common communication methods unicast and broadcast, and then discusses multicast.

Unicast is a method by which a host can send a packet of data to a single destination on the network. A unicast transmission is inherently point to point. In order to send the same data to many destinations, the host has to send a separate packet of the same data to each of the destinations. Thus, unicast uses several network data streams to communicate with more than one host; one to each host. This method is efficient only when one host needs to communicate with another host on the network.

Broadcast is a communication method which sends a packet of data to all the hosts on a subnetwork or network. A packet is sent only once and every host receives it as the packet is sent to a broadcast address. A broadcast packet uses up the network bandwidth as it travels over many networks. Broadcast is used to send out emergency information to all the hosts on the network.

Multicast is a transmission method which gives a host the ability to send out one packet to the network and have it received by a number of hosts (recipients) instead of sending out N different packets for N receivers. It conserves network bandwidth over unicast as the host sends out only one packet of data to the network and it can still be received by many hosts. The host sends the message on a multicast address, a special class of address (Class D) supported by IP for multicast. In Internet Protocol version 4 (IPv4) [8], Class D address covers the range of addresses from 224.0.0.0 to 239.255.255.255. Any number of hosts can listen on a given multicast address. Addresses from 224.0.0.0 to 224.0.0.255 are reserved for routing protocols to determine the topology of the network. For more information on the Class D address structure and routing protocols refer "IP multicasting: the complete guide to interactive corporate networks" [10]. There has to be multicast connectivity between the sender and the receivers. This means that the routers on the network must be enabled to route multicast traffic. Any number of hosts can announce their willingness to listen for messages transmitted on the multicast address. They do so by sending a message containing the multicast address of the desired group they want to join to their local multicast enabled routers. Routers exchange information using the reserved multicast addresses to keep themselves up-to-date with the list of the group members.

There are various routing algorithms that the routers use to ensure that the multicast packet is delivered to all the members of the group. Tree construction is a popular and efficient routing solution. A spanning tree defines a tree structure in which a single path connects any two routers on the network and is loop-free; messages are replicated only when the tree branches.

Routers construct a spanning tree, by exchanging information, through the network that reaches all the members of the multicast group. Whenever a router receives a multicast packet, it forwards the packet on all the links which belong to the spanning tree except the one on which the packet has arrived, ensuring that all the routers that serve a group member receive the packet. Figure 2.4 shows the path traversed by a multicast packet on the network.

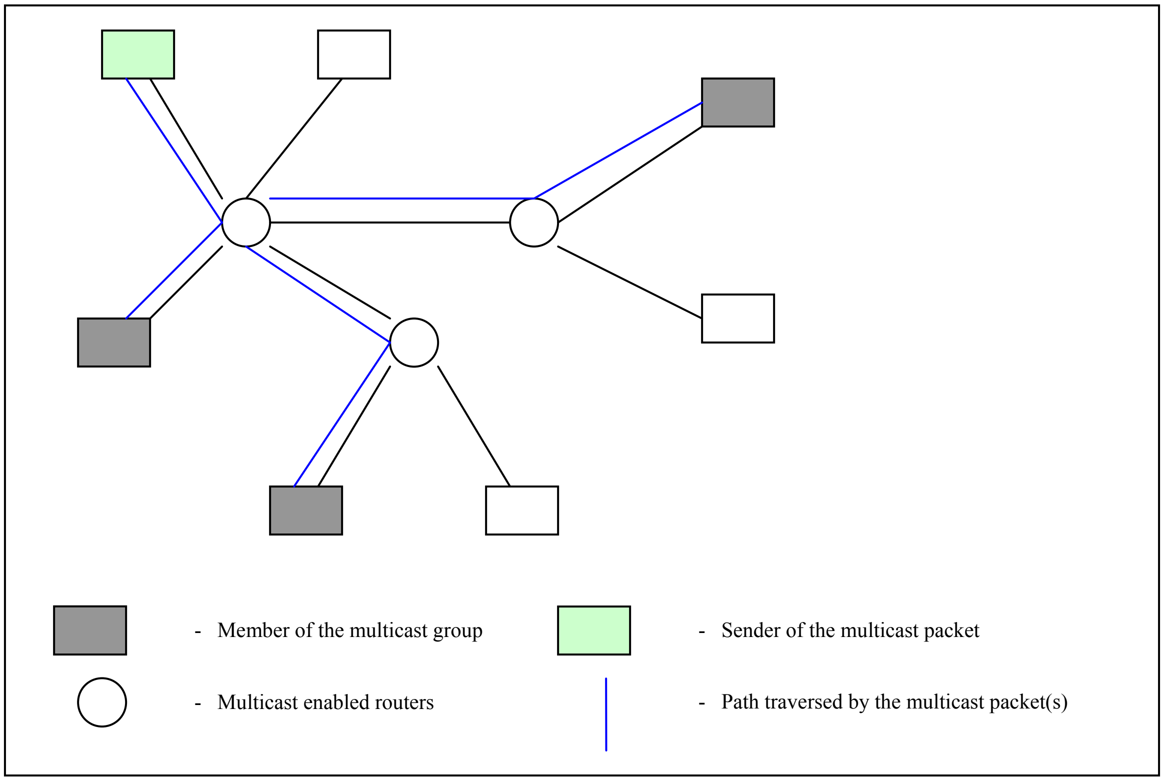

- Member of the multicast group
- Sender of the multicast packet
- Multicast enabled routers
- Path traversed by the multicast packet(s)

Figure 2.4: Multicast Packet Traversal

Routers forward the packet to the network that has hosts designated as members of the multicast group. Hosts can leave or join the multicast group at any time. If all the members of a multicast group on a particular network leave the group, the spanning tree is reconstructed so that the routers stops forwarding multicast data for that group to that network.

A multicast packet contains a special field called Time-To-Live (TTL). It is used to control the scope of a multicast packet. It determines how far a packet is allowed to travel in the network from the sending host. It is a hop count for the multicast packet. When a multicast enabled router receives a multicast packet, it examines the TTL of the packet and if it is 1, it will not forward the packet. If it is greater than 1, it will reduce the count by 1 and forward the packet appropriately. The range is from 0-255. A value of 0 limits the packet to the local host.

Multicast uses the User Datagram Protocol (UDP) [8], a transport protocol that uses the datagram approach to transfer packets from the source to the destination. Each datagram is an independent packet of data that carries the source and destination information in its header. It provides best-effort delivery and does not guarantee the delivery of the packet. Thus, multicast is an unreliable connectionless transfer of data from a source to one or more destinations. The applications using multicast have to ensure the reliability of the data transfer. Programs that use multicast in this thesis provide a very basic method of ensuring reliable data transfer.

As mentioned in Chapter 1, multicast is used to provide fault tolerance at the database level. As will be demonstrated later in the thesis, multicast is also used to locate the Web Service Registry.

2.3 Service Oriented Architecture Protocol (SOAP)

SOAP [1] is a protocol developed for Web Services to communicate with each other. SOAP uses eXtensible Markup Language (XML), a text based language, where any piece of information is encapsulated between an opening and closing tag (For example, <namePerson>xyz</namePerson>). The names of the tags are relevant to the piece of information they encapsulate. For example, a name of a person can be represented in XML as follows.

Name: John May

XML representation:

```
<name>
<first>John</first>
<last>May</last>
</name>
```

Due to this style of formatting, it is easier to parse XML data using XML parsers. An XML parser processes the XML formatted data and makes the information available to the program. More information on XML and XML parsing can be found in "Applied SOAP: Implementing .NET XML Web Services" [10].

As mentioned in the introduction, SOAP can be sent over any transport protocol. However, HTTP [4] is the most popular transport protocol used for SOAP. In the next section we look at the structure of a SOAP message and how a SOAP message is sent over a network using HTTP.

2.3.1 HTTP and SOAP

HTTP is a transport protocol widely used on the WWW to transfer data across the network. A web browser (client) and a web server communicate using HTTP. A HTTP message consists of a head and a body. The head gives information about the HTTP version being used, whether the message is a response or request message, and possibly other information. A message sent by the client requesting data from the server is termed as a request message and a message sent by the server containing the data requested by the client is called a response message. The data to be transferred is in the body of the HTTP message. A detailed structure of a HTTP message can be looked up in "Hypertext Transfer Protocol – HTTP/1.1" [4].

SOAP derives its message structure from HTTP. Likewise, a SOAP message appears within an envelope that has a body and a head (optional). The head of a SOAP message can be used to provide information useful for the authentication and routing of the message. The body of the message contains the actual programmatic information or data exchanged between the client program and the web service. SOAP messages are packed in the body of HTTP messages and sent over the network. Like a HTTP message, SOAP messages also can be categorized into a SOAP request and a SOAP response message. For the client and the web service to communicate, they should support HTTP and be able to interpret the SOAP message contained in the body of the HTTP message.

A web service can be viewed as a program having a set of remote functions (procedures) that can be independently called by the clients. Let's say a web service 'simpleService' is implemented having a function GetCurrency() that accepts a country's name as an input and outputs the country's currency. Thus, the prototype of the function could be defined as:

String GetCurrency(String country);

Both the input and the output arguments are String datatypes. This example web service is used to explain some of the concepts that follow.

Since SOAP is represented in XML, every piece of information in the SOAP message has an opening and a closing tag. The tags are named based on the conventions used by the SOAP protocol and also on the parameters and the functions of the web service.

A client can remotely invoke any function of the web service using a SOAP request message. The name of the function to be called, the input parameters to the function and other information is serialized by the client into a SOAP request message and sent to the web service in a HTTP request message. A SOAP request message sent by the client in a HTTP message to get USA's currency to the aforementioned web service is shown in Figure 2.5.

```
POST /soap/simpleService HTTP/1.1               |
Host: beagle.rnet.missouri.edu:8009             |
Content-Type: text/xml; charset=UTF-8           |
User-Agent: GLUE/1.2                            |    ←    HTTP Request Header
Connection: Keep-Alive                          |
SOAPAction: "GetCurrency"                       |
Content-Length: 481                            _|

<?xml version='1.0' encoding='UTF-8'?>               ←    XML Header        |
<soap:Envelope                        xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'                                |
xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'                   |
soap:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
        <soap:Body>                                                         |←HTTP Body
                <n:GetCurrency xmlns:n='http://tempuri.org/simpleService'>    (SOAP
                        <arg0 xsi:type='xsd:string'>USA</arg0>              |   Message)
                </n:GetCurrency>
        </soap:Body>                                                        |
</soap:Envelope>
```

Figure 2.5: SOAP Request Message

In addition to the HTTP Request header, a SOAP message requires an XML header to tell the data recipient that the data is represented using XML. The body part of the SOAP message contains the actual data. The body contains the name of the function and the input parameters to the function.

In the SOAP message, the Xmlns declarations are called namespaces. A namespace reference tells the location of the classes or the conventions used in the message. For example, xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/ tells the location, a URL, of the object or the style used for encoding the SOAP message. More information on the namespaces and other details about the SOAP message can be found in “Web Services Building Blocks for Distributed Systems” [1].

The web service 'simpleService' executes the GetCurrency() function and serializes the result 'Dollars' in a SOAP response message. The SOAP response message is embedded in a HTTP response message and sent to the client. The SOAP response message embedded in a HTTP message for the above request is shown in Figure 2.6.

```
HTTP/1.1 200 OK                                 |
Date: Wed, 22 Sep 2004 21:33:56 GMT             |
Content-Type: text/xml; charset=UTF-8           |   ← HTTP Response Header
Server: GLUE/1.2                                |
Content-Length: 505                             _|

<?xml version='1.0' encoding='UTF-8'?>       ←      XML Header                    |
<soap:Envelope                      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'                                           |
xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'                              |
<soap:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>                 ←HTTP
        <soap:Body>                                                             |   Body
                <n:GetCurrencyResponse xmlns:n='http://tempuri.org/simpleService'>  (SOAP)
                        <Result xsi:type='xsd:string'>Dollars</Result>          |
                </n:GetCurrencyResponse>
        </soap:Body>                                                            |
</soap:Envelope>                                                                _
```

Figure 2.6: SOAP Response Message

2.3.2 Web Services Description Language (WSDL)

In order for a client to call a function of a web service, it needs to know the name of the function, the input arguments to the function and their data types, the output of the function, and other information on how to bind to the web service. All this information is made available by the web service in the form a Web Service Description Language (WSDL) file. WSDL is an XML grammar used to describe a web service. WSDL encapsulates information between tags and hence can be easily interpreted by a client interested in using the web service. A WSDL file describes the characteristics of a web service. It acts as an interface for the client to access the functions of the web service. In some ways it is similar to the Interface Definition file generated for a DCE server.

The WSDL file that is generated for the aforementioned web service ‘simpleService’ is shown in Figure 2.7. The fourth last line in the figure gives the access point of the web service. The structure of an access point of a web service is explained later in this section. A detailed explanation of the tags and the information they encapsulate can be found in “Web Services Building Blocks for Distributed Systems” [1].

```
<?xml version="1.0" encoding="UTF-8" ?>
 <definition         name="simpleService"
   targetNamespace="http://www.themindelectric.com/wsdl/simpleService/"
   xmlns:tns="http://www.themindelectric.com/wsdl/simpleService/"
   xmlns:electric="http://www.themindelectric.com/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/">
   <message name="GetCurrency0SoapIn">
            <part name="arg0" type="xsd:string" />
   </message>
   <message name="GetCurrency0SoapOut">
            <part name="Result" type="xsd:string" />
   </message>
   <portType name="simpleServiceSoap">
            <operation name="GetCurrency" parameterOrder="arg0">
                 <input name="GetCurrency0SoapIn" message="tns:GetCurrency0SoapIn" />
                 <output name="GetCurrency0SoapOut"message="tns:GetCurrency0SoapOut" />
            </operation>
   </portType>
   <binding name="simpleServiceSoap" type="tns:simpleServiceSoap">
            <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
            <operation name="GetCurrency">
   </binding>
   <service name="simpleService">
        <documentation>simpleService web service</documentation>
        <port name="simpleServiceSoap" binding="tns:simpleServiceSoap">
            <soap:address location="http://beagle.rnet.missouri.edu:8009/soap/simpleService" />
        </port>
 </service>
</definitions>
```

Figure 2.7: WSDL Representation of a Web Service

2.3.3 Requirements to Host a Web Service

The server (machine) must have some basic components in order to be able to run a web service. Figure 2.8 shows the components needed on a server (machine) to host a web service. To use HTTP as a transport protocol, a web service needs to be running a HTTP web server to listen for SOAP request messages embedded in HTTP request messages. Upon receiving a HTTP request message, the Listener retrieves the SOAP message from the body of the HTTP message and passes it to the XML parser. The XML parser retrieves information from the body of the SOAP message regarding which function is to be executed and what are the input parameters. After the information has been retrieved, a dispatcher is called that converts the input parameters received to the native data representation and invokes the required function. After the function is executed, the output is serialized in a SOAP message by a SOAP Serializer and sent to the HTTP server to send it to the client. Thus the core components required on a server (machine) to run a web service are the HTTP server, the XML parser, the SOAP Serializer and the dispatcher. These can be grouped together under one entity called a SOAP Factory (SF). One of the other responsibilities of a SF is to generate a WSDL file describing the web service. This WSDL file is read by the client prior to calling a function of the web service.

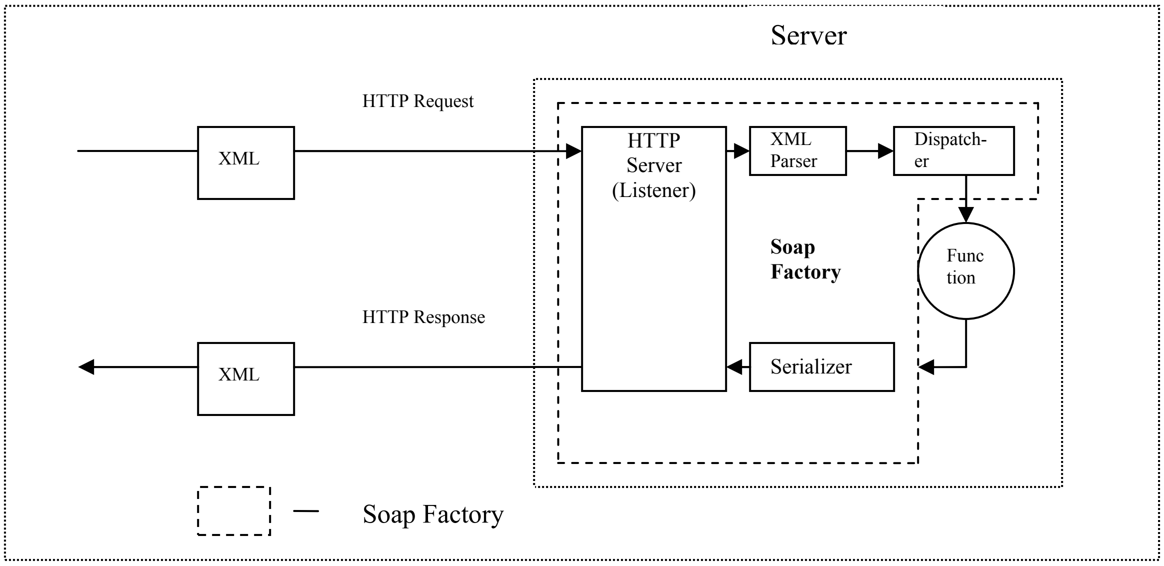


Figure 2.8: Soap Factory on the Host Machine

As mentioned earlier, a web service requires a HTTP server to listen and accept SOAP messages. The access point (URL) of the web service consists of the address of the HTTP server and the web service name. The address of the HTTP server is comprised of the DNS name of the server, the port on which the HTTP server is started and the path where the server is listening for the SOAP messages. Thus, the prototype of the access point (URL) of a web service looks like

http://{server}:{port}/{path}/{Service Name}.wsdl

The web service 'simpleService' is started on a server beagle.rnet.missouri.edu. The access point (URL) of the web service 'simpleService' is http://beagle.rnet.missouri.edu:8009/soap/simpleService.wsdl where beagle.rnet.missouri.edu is the machine on which the web service is running, 8009 is the port number and /soap is the path where the HTTP server is listening for incoming messages. Two services cannot be started on the same port.

2.3.4 Requirements to Access a Web Service

A client wanting to call a web service should have the necessary components in order to send a SOAP message to the web service. Figure 2.9 shows the components needed on a client machine to access a web service. A client requires an XML parser to parse the WSDL file to retrieve information about a web service. After the WSDL file has been interpreted, a proxy generator creates a proxy to be used by the client program to make remote function calls to the web service. A proxy is a native language interface of the web service function, like a DCE client stub, that makes the remote calls to the web service on behalf of the client program. It also channels the results received in the SOAP response from the web service back to the client program. With the help of a proxy, the program can make remote function calls as if the functions were residing on the local machine. A SOAP serializer generates the SOAP request message to be sent to the server. Thus, the XML parser, the SOAP Serializer and the proxy generator are the essential components of a Soap Factory (SF) on the client.

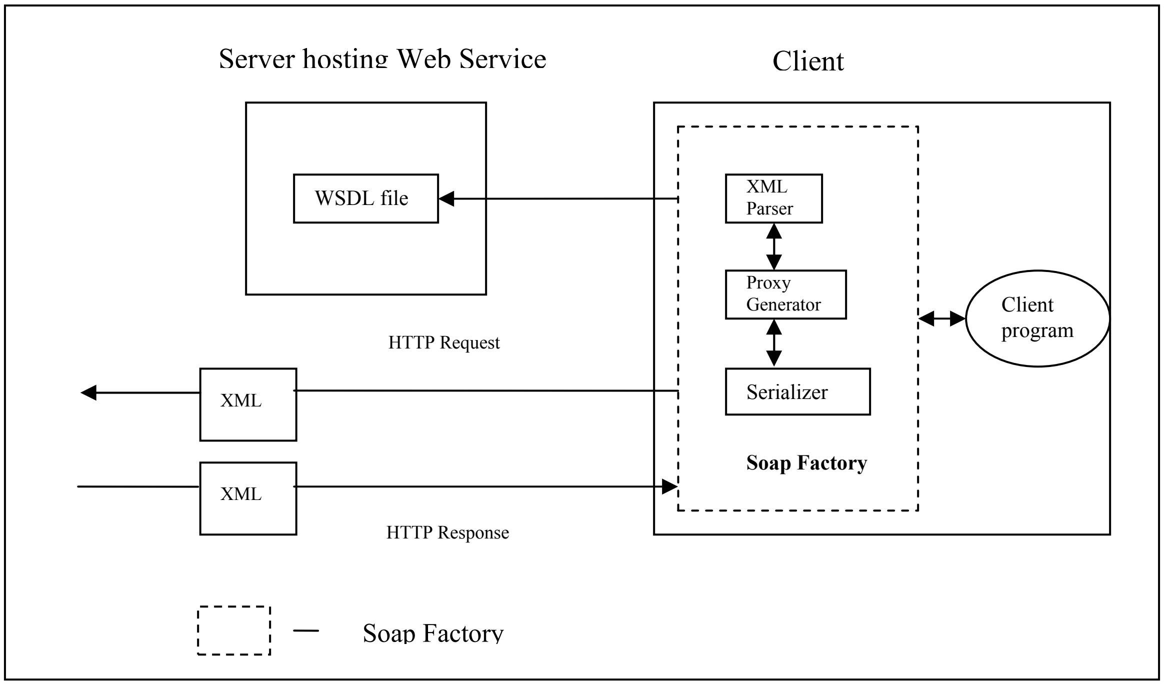


Figure 2.9: Soap Factory on the Client Machine

There are several SF toolkits available in the industry to build and access web services. Different SF toolkits support different languages and platforms. To name two, Microsoft uses .NET and Sun Microsystems uses J2EE. The SF toolkit used to build web services for this thesis is called GLUE provided by "webmethods.com" [11]. It provides the necessary tools to deploy and access web services using JAVA. The steps involved in creating, deploying and accessing a web service using GLUE can be found in "Web Services Building Blocks for Distributed Systems" [1].

2.3.5 Discovery of a Web Service

The main purpose of inventing SOAP is to introduce a protocol that can be understood by any machine or platform and, hence make web services interoperable and accessible by a larger group of clientele. Thus, SOAP is intended to overcome the communication barriers imposed by heterogeneous machines on programs (services). Even though SOAP enables a high degree of interoperability, the web services offered by providers still need to be discovered by clients for them to utilize the web services. Clients need to know the location of the web service and its WSDL file. This identifies the need to have a well-known Web Service Registry (WSR) that maintains information about web services registered with it. Clients, once they know the location of the WSR, can search for web services based on their requirements. After retrieving the location of the web service and its WSDL file from the WSR, they can make remote function calls to the web service from their programs. Other information about the web service such as its owner, a web URL to the documentation page of the web service, and other information can be maintained in the WSR as well. Thus, a WSR provides an interface for users to publish, search and retrieve information about a web service.

The interface of the WSR could be in the form of web pages where users can type in information in a web browser and have the results displayed to them in the form of web pages. The interface of the WSR could also be a web service which implements the WSR operations as functions. These functions can be invoked programmatically from a client program using SOAP. There are several public registries maintained by IBM [16], Microsoft [17], and others to facilitate discovery of web services.

This thesis focuses on building a WSR having a web interface and a web service interface. Both the interfaces are designed to be fault tolerant. The web interface of the WSR provides the access point(s) of the web service interface of the WSR whereas a multicast client program is used to retrieve the address(s) of the web interface of the WSR. The WSR is covered in detail in Chapter 5.

2.3.6 Deploying, Locating and Invoking a Web Service

After having introduced SOAP and the concepts related to web services, we conclude the discussion with an overview of the process followed to deploy, locate, and invoke a fault tolerant web service. Figure 2.10 shows the steps involved in the process.

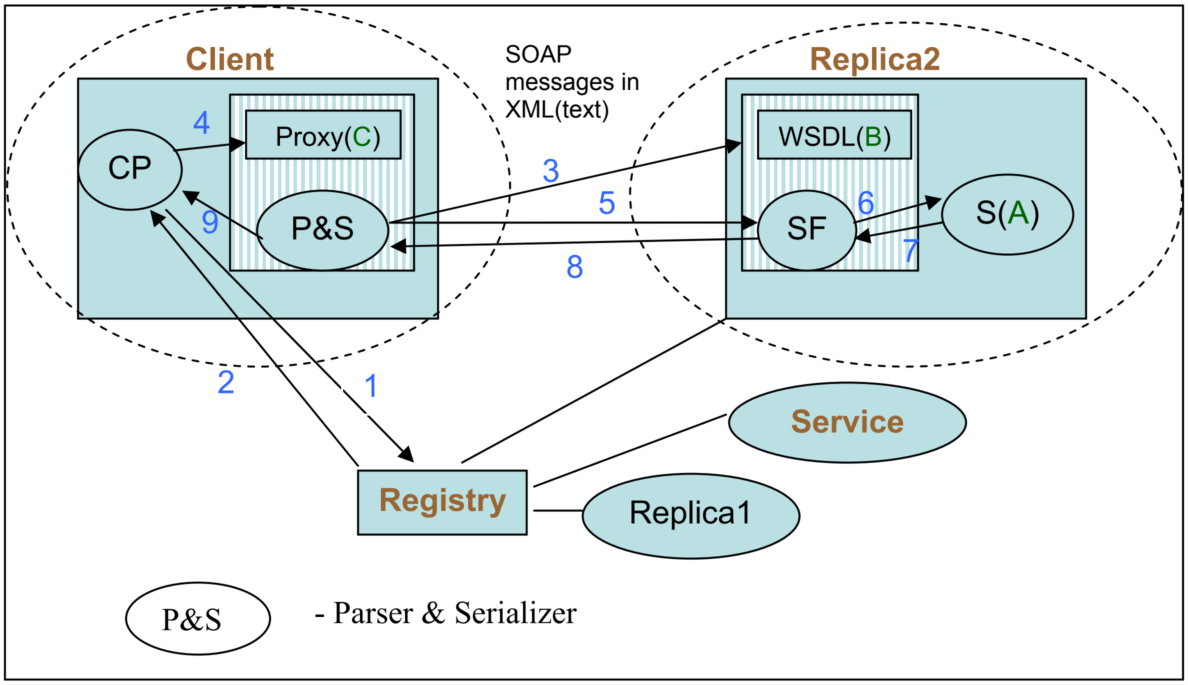


Figure 2.10: Deploying, Locating and Accessing a Web Service

Deploying a Web Service

A program (S) shown as [A] in Figure 2.10, is built using any programming language supported by the development environment. The program implements remote procedures (functions) that can be remotely called by clients. The Soap Factory (SF) reads the native language interface of the program and generates a unique WSDL file shown as [B] by mapping the native representation of the program to its associated XML schema. Thus, the program is converted into a web service with the help of a Soap Factory. The Soap Factory runs the HTTP server necessary to listen for the incoming SOAP messages from the clients. Figure 2.10 shows that the web service is deployed on three servers with one instance of the web service addressed as **Service** and the other two as **Replica1** and **Replica2**.

Locating a Web Service

A Web Service **Registry** (WSR) is maintained that has the access points of the three copies of the web service. A **Client** searches the WSR based on some criteria such as the names of the functions or some keywords [step 1]. If a web service is found that matches the criteria, the WSR returns the access points of any one or all instances of the web service to the client [step 2]. The advantage of sending access points of all the instances of the web service to the client is that the client can contact instances on other access points if either one happens to fail. In the future, the client contacts the WSR to get new access point(s) only if all three instances are not available. The Soap Factory on the client reads the WSDL file of Replica2 [step 3] and generates a local proxy shown as [C] to be used by the client program for calling the remote procedures of the web service. A proxy is a native language interface having similar procedure calls as that of the remote web service and can be termed as the client stub. Thus, the Soap Factory on the client maps the XML schema of the WSDL file to the native language representation of the client program (CP).

Invoking a Web Service

The client program calls one of the remote procedures (functions) of the web service by calling the proxy [step 4]. The proxy converts the data from the native language representation of the program to the XML representation, packs the data in the SOAP request message using the serializer and sends it to the web service [step 5]. The Soap Factory on the server accepts the incoming SOAP request sent by the client through the HTTP server. The Soap Factory on the server translates the data from the XML representation to the representation understood by the program and forwards the request to the program [step 6]. The program executes the necessary function and sends the output to the Soap Factory [step 7]. The Soap Factory serializes the data received from the program in an XML representation, forms a SOAP response message and sends it to the SF on the client [step 8]. The Soap Factory on the client parses the SOAP response received from the web service and sends the output to the client program [step 9].

This chapter discussed the two protocols DCE-RPC and SOAP, and also the multicast transmission method. It introduced the concepts and described the interaction among the programs using these protocols. The next chapter (Chapter 3) discusses the design of the fault-tolerant system used to start and monitor multiple instances of a web service. The fault-tolerant system is comprised of components (programs) that use DCE-RPC, SOAP and multicast. The next chapter introduces the components of the system and discusses the role played by each component in the system.

3. Design of the System to Administer Fault-Tolerant Web Services Using DCE-RPC and Multicast

The system to administer and monitor multiple instances of a web service is comprised of the basic components, such as the Web Service Manager, the fault manager and the database. In addition to the basic components, the design of the system includes other programs that help the basic components communicate with each other and also with the instances of a web service. This chapter discusses the various components of the design and the deployment of the components on different servers (machines) leading to a fault tolerant system to administer fault-tolerant web services.

3.1 Design

The discussion begins with a block diagram of the system design shown in Figure 3.1. It shows the basic components of the system, such as the Web Service Manger, the fault manager and the database. The figure shows two instances of each basic component deployed on two servers (machines), marked as (A) and (B) to avoid a single point of failure. It also shows a cluster of web service instances deployed on different servers (machines). A cluster of a web service is comprised of all instances of the same web service residing on different servers (machines). The cluster can contain as many instances of the web service as needed. In Figure 3.1 two instances (A) and (B) of a web service form a cluster that needs to be administered.

The administration of a cluster includes starting the multiple instances of a web service on different servers (machines), checking their availability periodically and occasionally starting a new instance in response to the failure of a running instance. Each web service cluster is given a universal unique identifier (UUID) and, all instances belonging to the same cluster have the same identifier.

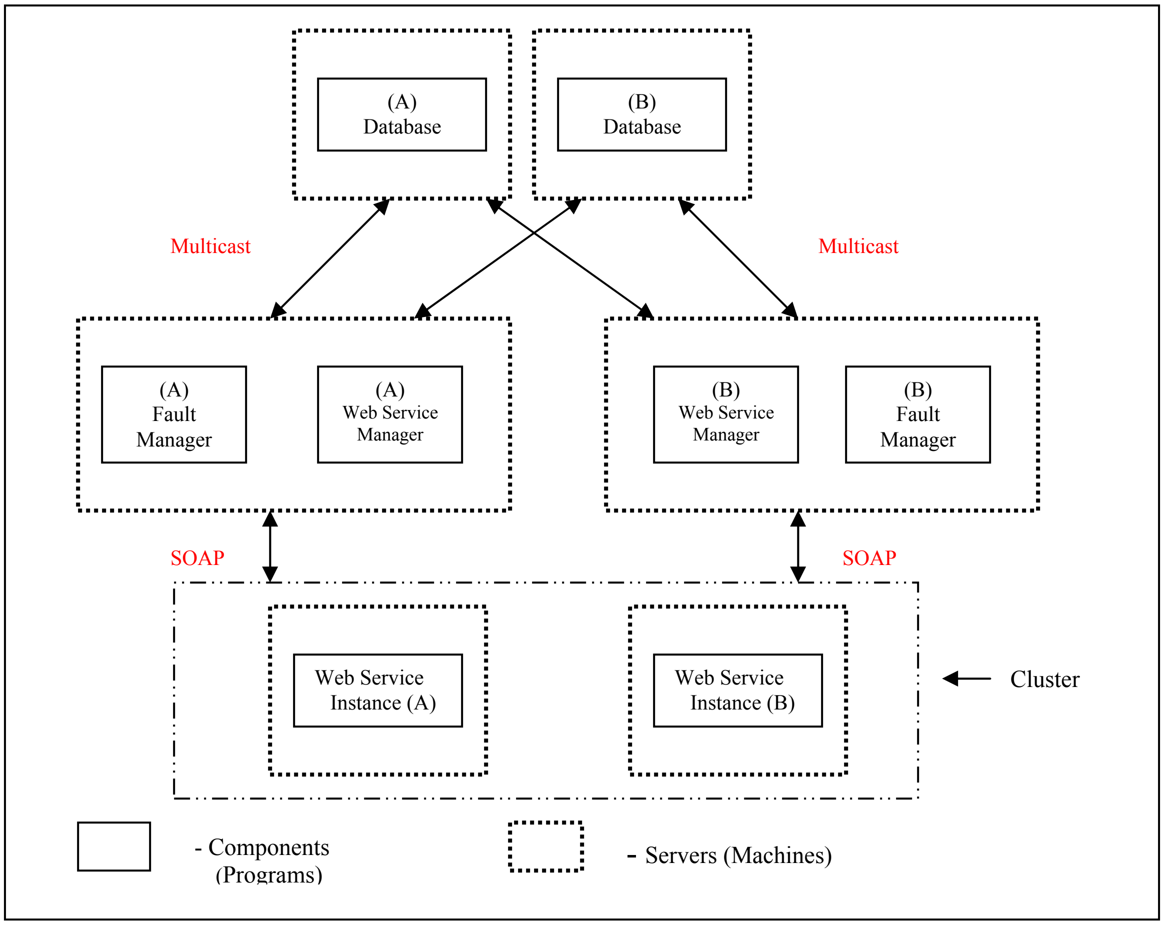


Figure 3.1: The Basic Block Diagram of the Design

Additional components (programs) are introduced, as the design is discussed further and consequently the block diagram becomes more detailed. The arrows in Figure 3.1 indicate communication among the components. As shown in the diagram, the managers (Web Service Manager and fault manager) communicate with the database using multicast. The managers use SOAP to interact with the web service instance(s).

3.2 Web Service Manager

The Web Service Manager (WSM) is the core component of the system. It provides the necessary interface to administer a cluster of a web service. The Web Service Manager implemented as a DCE server provides the interface needed to administer a web service cluster as a set of procedures. These procedures can be remotely called by a DCE client program. A DCE client program, called Wsmclient, communicates with the Web Service Manager. Wsmclient is discussed in Section 3.8 of this chapter. The DCE client program provides an interface for the user (administrator) to call the Web Service Manager's remote procedures to start and administer a web service cluster.

The Web Service Manager implements procedures to start a web service cluster and occasionally start a new instance of a web service on a back-up server (machine) when it receives a notification of a failure of any of the running instances in the cluster. The necessary information about a web service cluster is maintained and updated in the Oracle database by the Web Service Manager using multicast. The information maintained in the database consists of the UUID given to the web service cluster, the names of the servers (machines) where the instances are running, the number of instances running in the cluster, the access points of the instances and the addresses of the back-up servers (machines) where a new instance can be started in the event of failure of any of the running instances. The schema of the database design is discussed in Appendix A. Upon receiving a notification of a failure of any of the instances in the cluster, the Web Service Manager obtains the address of the back-up server (machine) from the Oracle database and starts a new instance on the back-up server (machine). The WSM also provides a set of procedures to determine the status of any instance in the cluster, to stop any instance and to obtain information about a given web service cluster.

The Web Service Manager is implemented as a DCE server called Wsms. Wsms must be deployed on more than one server (machine) to provide fault-tolerance. The source code for implementing the Web Service Manager as a DCE server can be found in Appendix B. In Figure 3.1, the Web Service Manager is shown running on two servers (machines). The Web Service Manager can be started as a part of the server's (machine) boot-up process such that whenever the server is booted the Web Service Manager starts automatically. When the Wsms DCE server starts execution, it registers itself with the CDS so that a DCE client program can contact the CDS to get the address of the host machine running the Wsms DCE server as explained in Chapter 2.

In order to start a web service instance on a different server (machine), the Web Service Manager needs to communicate with the specific server (machine) where the web service instance is to be started. This requires that a program must be running on that remote server (machine) for the Web Service Manager to communicate with. In this scenario, the Web Service Manager, a DCE server, acts as a client to this program on the remote server (machine). The program on the remote server performs the necessary operations on behalf of the Web Service Manager, such as starting an instance or stopping an instance upon receiving a request from the Web Service Manager.

The Web Service Manager and the program on the remote server (machine) can either use the DCE-RPC network protocol or SOAP to communicate with each other. If the DCE-RPC network protocol is chosen then the program on the remote server (machine) needs to run as a DCE server. To run a DCE server, the required DCE software is needed to be installed on the server (machine). On the contrary, a web service is comparatively easier to implement since it can be built using the Soap Factory tools which are easy to install and maintain. Thus, the program on the remote server (machine) that the Web Service Manager communicates with is chosen to be a web service. This prevents any stringent requirements like the installation of DCE components on the servers (machines) that run the web service instances. The web service on the remote server that the WSM communicates with is called the GenericSoapServer. The Web Service Manager communicates with the GenericSoapServer to start and stop a web service instance on that server (machine). The functions implemented by the GenericSoapServer and the parameters needed to be supplied to the functions to start or stop an instance are explained later in Section 3.5 of this chapter. Like the Web Service Manager, the GenericSoapServer can be started as a part of a server's (machine) boot-up process. The GenericSoapServer needs to be started on servers (machines) where the web service instances are to be started.

The DCE-RPC natively supports the C Language and hence, the Web Service Manager (Wsms) is written in C Language. As mentioned in Chapter 2, this thesis uses the Soap Factory GLUE, which has libraries built in JAVA to develop and access web services. Hence, the web service GenericSoapServer is written in JAVA.

In this section along with the explanation of the Web Service Manager, two programs were introduced. One is Wsmclient, which provides the interface for the user to communicate with the Web Service Manager. The other is the GenericSoapServer, which helps the WSM in communicating with the servers (machines) where the web service instances need to be started. The basic block diagram with the addition of these programs is shown in Figure 3.2. The diagram shows one more program called Listener which is covered in the following section. The following discusses the interaction between the Oracle database and other components of the fault-tolerant system.

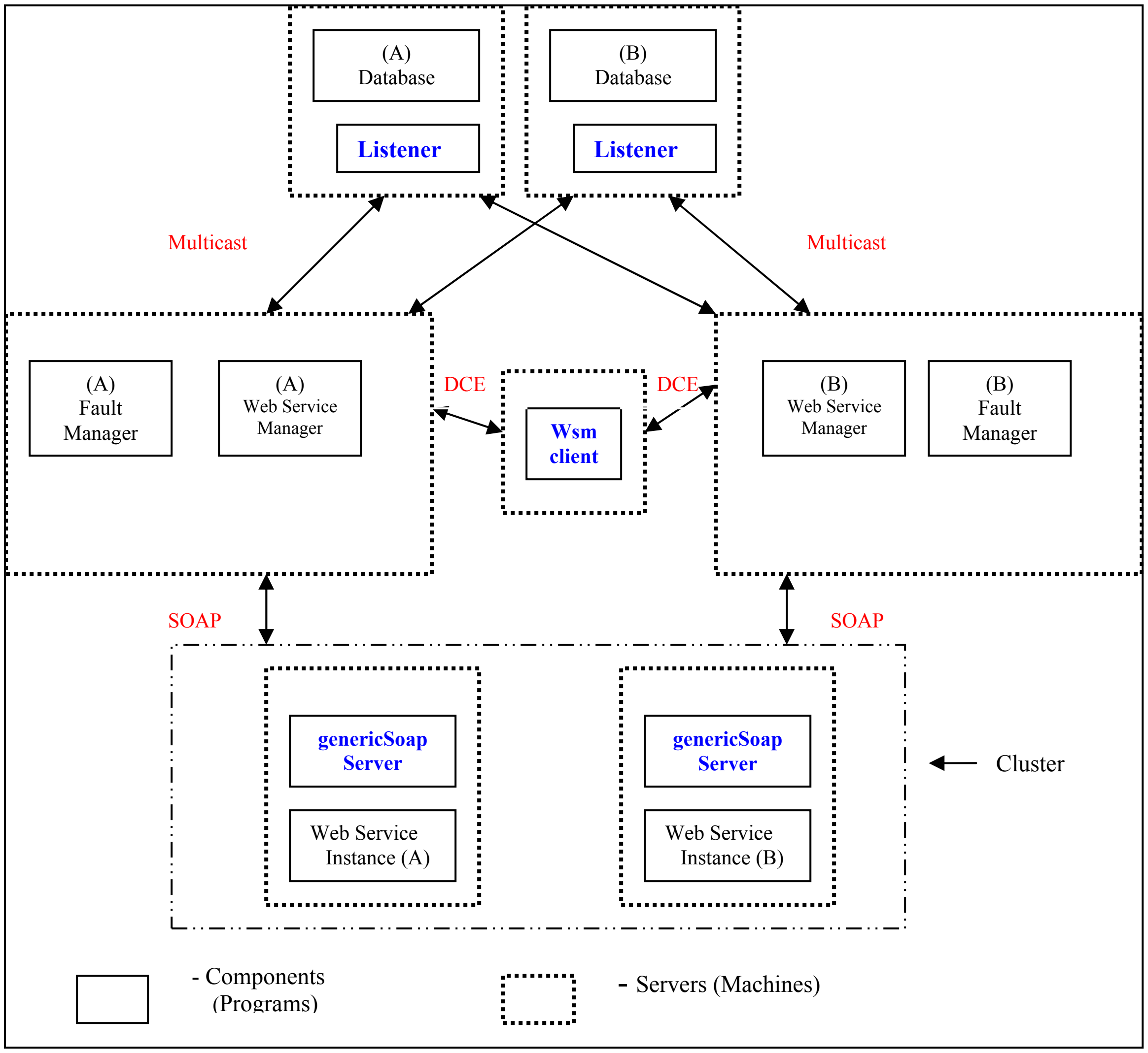


Figure 3.2: The Block Diagram of the Design with the Introduction of Wsmclient, GenericSoapServer and Listener

3.3 Database

A database is an organized collection of data that can be easily and efficiently searched, retrieved and updated. The database can be thought of as a program that stores data and provides an interface to other application programs to access the stored data. Various state-of-the-art products are available in the industry that can be used to create a database for a specific purpose. The choice of the database system depends upon the volume of the data to be stored, the level of abstraction the software offers to the applications in searching and retrieving the data, and the efficiency of the database in utilizing the space to store the data. One of the important concerns with the database is its accessibility from the remote machines. Some databases can be easily connected from remote machines while the remote connection requirements for others are complicated. For this thesis, a database is needed that can be remotely connected from another machine. Oracle was chosen primarily because of the need of the project and secondarily because it was already available to the machines where the fault-tolerant system was built and tested.

Fault-tolerance at the database level is provided by having multiple instances of the database on different servers (machines). This is shown in Figure 3.2. Any program that needs to update data in the database has to communicate the update to all the instances of the Oracle database to maintain consistency of the data among the instances. The program can update all the instances of the database by communicating with the instances one at a time, but this is not an efficient way of doing it. An alternative and efficient way is to send the update request to all the instances of the database simultaneously. This is done with the help of multicast. An update request is sent on a multicast address such that all the listeners listening on a given multicast address receive the update request. In order to support the use of multicast in the fault-tolerant system, multicast listeners are installed on the servers (machines) where the database instances are deployed. The multicast listener is called Listener. This is shown in Figure 3.2. The Listener listens for the update requests and updates the local instance of the database. The source code of Listener can be found in Appendix B. The Listener programs running on different servers (machines) listen on the same multicast address. A Listener can be started as a part of the server's (machine) boot-up process.

Multicast is used within the system to communicate between the database instances. The other components within the system, however, interact with each other using DCE-RPC, and HTTP and SOAP protocols. Thus, for the fault-tolerant system to be operable, only the database instances are required to be deployed on a multicast-enabled network

As stated in Chapter 2, multicast is an unreliable method of communication and hence, the applications using multicast have to ensure reliable transfer of the messages between the sender and the recipients. This thesis adopts the following strategy to ensure reliable data transfer between the sender and the recipients of the messages.

The client program (sender) sends a multicast message on the multicast address and waits to hear a response back from the recipients of the message. The recipients upon receiving the multicast message retrieve the address of the sender from the multicast message. All the recipients send a unicast message back to the sender using the address of the sender retrieved from the multicast message, indicating the message was received by the recipients. Thus, the communication from the sender to the recipients of the message is done using multicast whereas the recipients send a response back only to the sender using unicast. The sender upon receiving the first response back from any one of the recipients comes out of the waiting loop and proceeds further. If the sender doesn't receive a response from any one of the recipients, it continues to send the message to the recipients at regular intervals for a specified number of times until it hears a response back. The number of times the message is re-sent is set to 3, but it can be changed if needed. If the client program does not get any response after having sent the multicast message for the specified number of times, it reports a failure. A client not getting a response back even after sending the multicast message a number of times indicates that the network between the client and the Listener(s) is broken or the Listener(s) are not running. A notification of such a failure is sent to the administrator.

The two basic database operations are the read and the update operations. The read operation does not modify the data in the database whereas an update operation modifies some data in the database. While doing a read operation, only one instance of the database needs to be contacted whereas an update operation requires all the instances of the database to be contacted and updated. The Listener program has to accommodate both types of operations. The role played by the Listener to execute the read and the update operations is discussed below.

3.3.1 Read Operation and Update Operation

In order to perform a read operation, the application program (client) needs to connect to the database instance and needs to know the location of the database instance. The database instance could be on the same server (machine) as the application program or on a remote server (machine). The address (location) of the database can be hard-coded in the application program or the application program can fetch the address during runtime to communicate with the database. As mentioned earlier, this thesis uses multiple instances of the same database running on different servers (machines) to achieve fault-tolerance and whose addresses are not known beforehand. A new database instance can be started on any server. Thus it is desirable that the address of the database instances not be hard-coded in the application program (client). The application program gets the address of the database instance(s) during runtime by communicating with the Listener(s) using multicast. The Listener(s) knows the address of the local database instance. When contacted by the application program the Listener sends the address of the database instance to the application program. The application program uses this address to remotely connect to the database instance.

Figure 3.3 shows three database instances (A), (B) and (C). The application program (client) sends a multicast message to the Listener(s) requesting the address of the database instance(s) [step 1]. The Listener(s) running on different server(s) sends the address of the local database instance back to the client using unicast[step 2]. The client program selects the first address received to remotely connect to that database instance and execute the read operation [step 3].

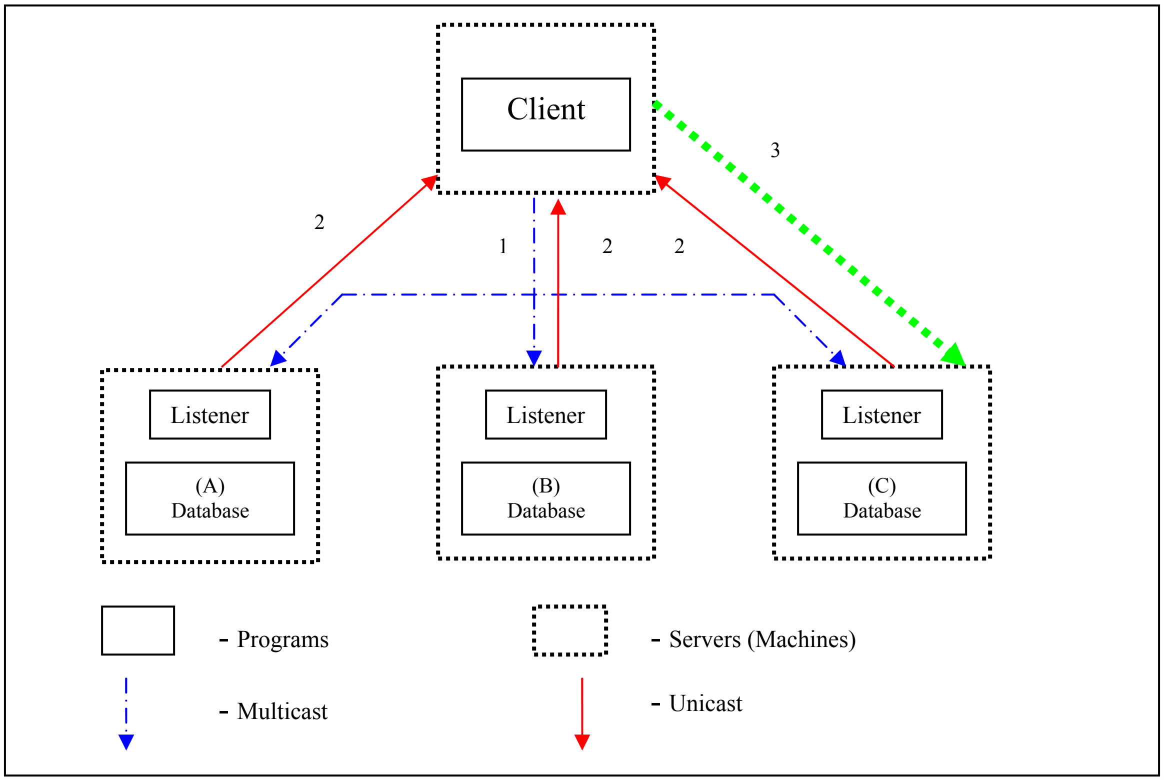


Figure 3.3: Read Operation

The application program tries to get the address(s) of the database instance(s) when it starts execution and, if it cannot find any database instance(s) it aborts reporting a "database not found" error. After having found a database instance to connect to, if the application program is not able to connect to that database instance, it reissues the multicast message to the Listener(s) to get possibly new address(s) of the database instance(s).

For an update operation all the instances need to be contacted. As shown in Figure 3.4, the client program forms the update request that needs to be executed on the database instances. It then sends a multicast message containing the request on the multicast address to the three instances (A), (B) and (C) and waits for a response [step 1]. The Listener(s) receives the multicast message sent by the client. With the help of the interface provided by the database, the Listener executes the update request on the local instance of the database [step 2]. The Listener(s) sends the status of the execution of the request back to the client using unicast. If the update is successfully executed it sends an "OK" response; otherwise an error message is issued [step 3].

The client program comes out of the waiting loop as soon as it receives the first "OK" response from any of the Listeners. It is possible that instance A and instance B got updated correctly while the multicast update request sent by the client did not reach instance C. This leaves instance C in an inconsistent state. The action taken when a database instance falls in an inconsistent state compared to the other database instance(s) is explained in Section 3.3.3 of this chapter.

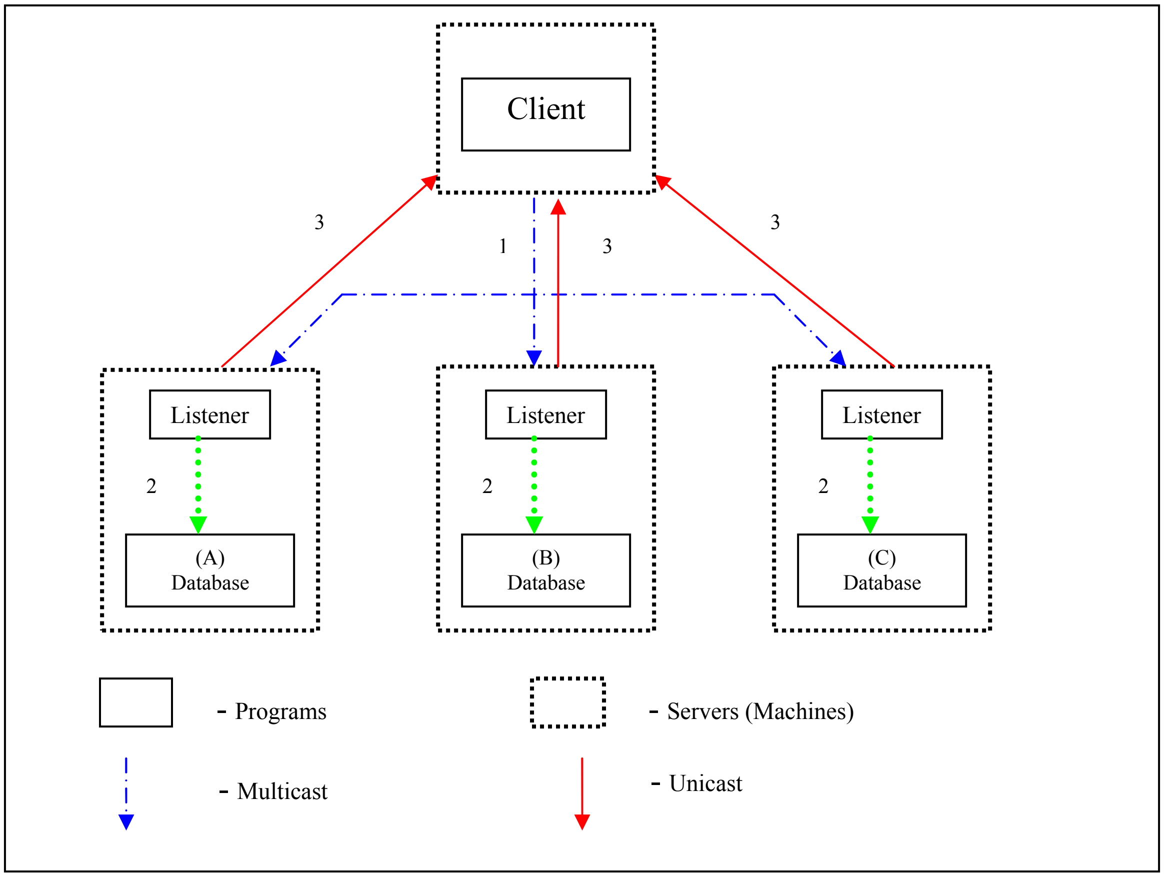


Figure 3.4: Update Operation

The Listener, which is written in C Language, connects and executes the update on the Oracle database with the help of a tool called pro-C/C++ precompiler supplied by Oracle. A complete guide to the precompiler can be found on the Oracle website [13].

3.3.2 Authenticating Update Request and Logging to Prevent Duplicate Updates

An update message is sent using multicast and it contains the update request to be executed to update data in the database. The request can have confidential information such as the administrator's or user's password or his personal information. Hence, it is required to encrypt the update request message before it is sent to the network to avoid interception of data on the network. All the update request messages are encrypted before they are sent on the multicast address and decrypted by the Listeners. Also, it is important to verify the source (sender) of the update request message to ensure that the update message received is legitimate. This is achieved by providing the Listener with a list of addresses of the hosts (machines) that would possibly be sending update request messages. Upon receiving an update request message, the Listener retrieves the address of the sender and verifies the identity by referring to the list of addresses. If the address of the sender belongs to the list, the update request message is considered to be legitimate; otherwise the message is discarded.

As discussed earlier, the sender of the multicast message continues to send the message for a specified number of times until it hears a response back from any of the recipients. Depending upon the traffic in the network or other reasons, it is quite possible that the response sent by the Listener (recipient) gets lost and the client reissues the multicast message. This is tolerable if the re-issued multicast message is to retrieve the address of the database instance, but it is not acceptable if the multicast message is to execute an update request on the database. The update could be performed twice and might leave the data in an inconsistent state. Hence, precaution is required on the Listener's side to ensure that the same update query is not executed twice.

The update request message carries a transaction identifier, which is unique for each update message sent. The sender of the update message ties a transaction identifier with the update message. The Listener logs the update request in the transaction table along with the transaction identifier. The transaction table contains information about the update transactions performed on the database instance. The schema of the transaction table can be found in Appendix A. Before any update request is executed, the transaction table is checked to see if the request has already been executed with the help of the unique transaction identifier. If an entry with the same transaction identifier is found in the transaction table, the request is not executed again. If no entry is found, the update request is executed. This logging mechanism with the help of transaction identifier ensures that the update query is executed only once.

3.3.3 Database Synchronization and Refresh

Update requests sent by the clients in a multicast message are observed by only those database instances that are available on the network at that instant. Any new database instance that has just started or a database instance recovering from a failure is not aware of the prior update requests sent by the clients. Synchronizing a new database instance or a recovering database instance with the other database instances that have seen all the update requests is critical to the fault-tolerance of the system as a whole.

The Listener program that listens for the update multicast messages implements a procedure to synchronize the local database instance with the other available database instances. The Listener tries to synchronize the local database instance with other instances when the Listener is started on a server (machine) or in other words, when a local database instance is made available to the clients. The procedure implemented by the Listener to synchronize the local database instance when it is started is shown in Figure 3.5. Figure 3.5 shows the steps executed by the Listener to synchronize the local database instance with the other database instances available on the network.

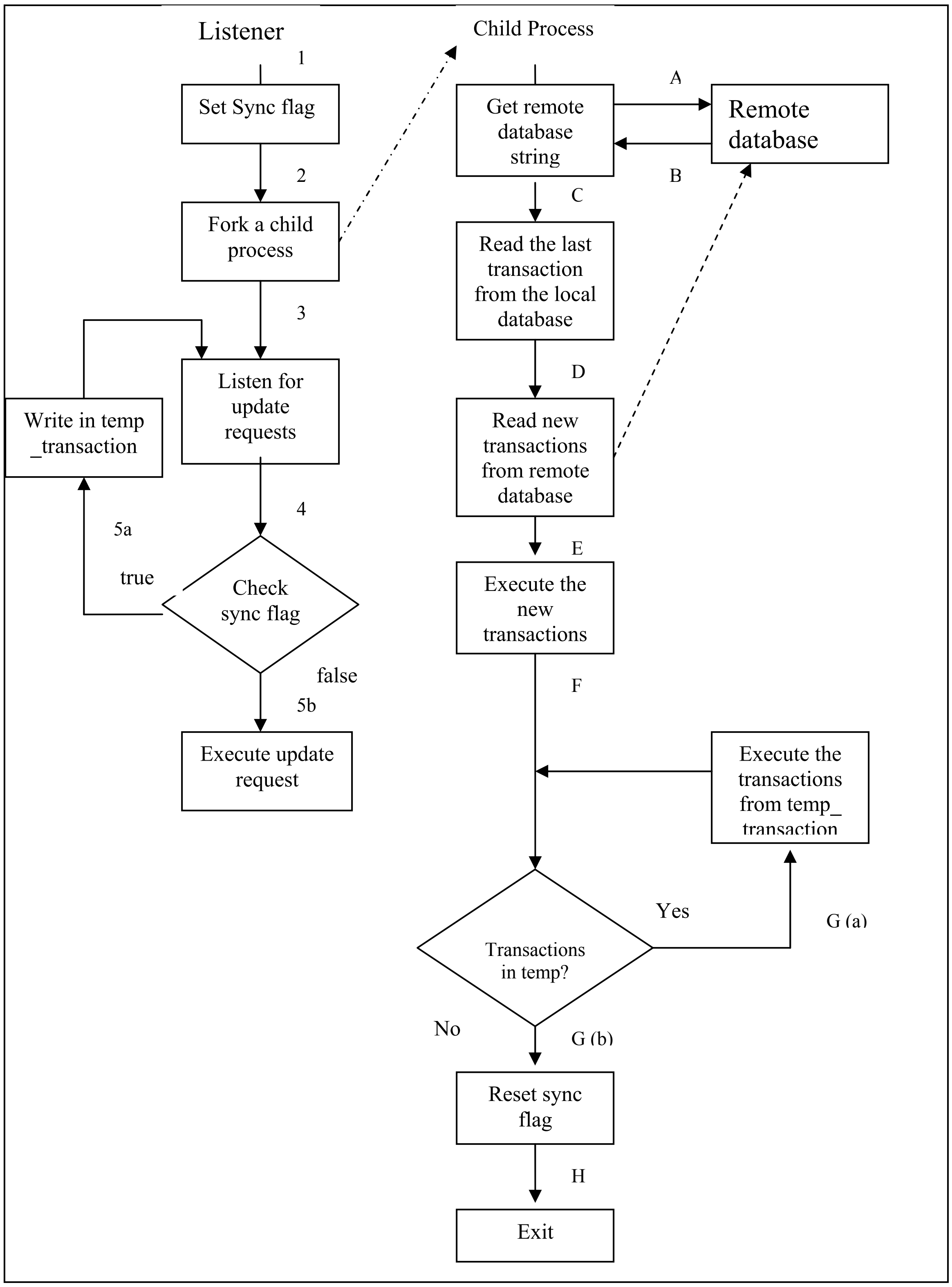

Listener
Child Process
1
Set Sync flag
2
Fork a child process
3
Listen for update requests
Write in temp _transaction
4
5a
true
Check sync flag
false
5b
Execute update request
Get remote database string
A
B
Remote database
C
Read the last transaction from the local database
D
Read new transactions from remote database
E
Execute the new transactions
F
Execute the transactions from temp_ transaction
Transactions in temp?
Yes
G (a)
No
G (b)
Reset sync flag
H
Exit

Figure 3.5: Database Synchronization

The two new entities used in the synchronization procedure besides the Transaction table that stores the transactions are the *Temp_transaction* table and the *Database_Sync* flag. The schema of the Temp_transaction table and the Database_Sync flag can be found in Appendix A. The Transaction table stores all the transactions received by a database instance up to the present time. It stores the timestamp, the update request and a unique transaction identifier. The Temp_transaction table is used to store the update requests that are sent by the clients when the database synchronization or the database refresh is taking place. Database refresh is explained later in this section after Database Synchronization.

An update request stored in the Temp_transaction is not performed (executed) on the local database instance until the database synchronization or database refresh has completed. The Database_Sync flag is used to indicate whether or not the database synchronization or database refresh has completed. If Database_Sync flag is set to 'true', it means that the synchronization or refresh is taking place. If it is set to 'false' it indicates that the local database instance has been synchronized or refreshed with the other database instances.

As shown in Figure 3.5, the Listener sets the Database_Sync flag when it is started [step 1]. After it sets the Database_Sync flag, the Listener forks a child process that executes the steps to synchronize the local database instance [Step 2]. The Listener and the child process execute simultaneously. After forking the child process, the Listener starts to listen for the update request messages on the multicast address [step 3]. A child process is forked by the Listener to do database synchronization so that the Listener can listen to the update requests while the database synchronization is taking place simultaneously. If the Listener happens to receive any update requests it checks the Database_Sync flag [step 4]. If the flag is still set to 'true' it indicates that the child process has not finished synchronizing the database and the Listener writes the update requests in the Temp_transaction table [step 5a]. When the database synchronization is taking place, the Listener only receives requests on the multicast address. It does not respond to any of the requests received. The database instance is not seen by the clients until the local database instance has been synchronized.

In order to synchronize the local database instance, the child process needs to know the location of a remote database instance that has seen all the transactions to the present. The child process sends out a database address request message on the multicast address [step A]. The database instance(s) running on the other servers (machines) respond to this message with a unicast message containing the database address to connect to that remote database instance [step B]. The child process selects the first response received and uses the database address to remotely connect to the database instance. The child process reads the transaction identifier of the last update request that was executed on the local database instance from the Transaction table [step C]. The child process connects to the remote database instance and reads the new transactions that have been executed on the remote database instance [step D]. Since the same transaction identifiers are received by all the database instances for the same update requests, it is possible to determine the set of transactions that need to be imported from the remote database instance to synchronize the local database instance.

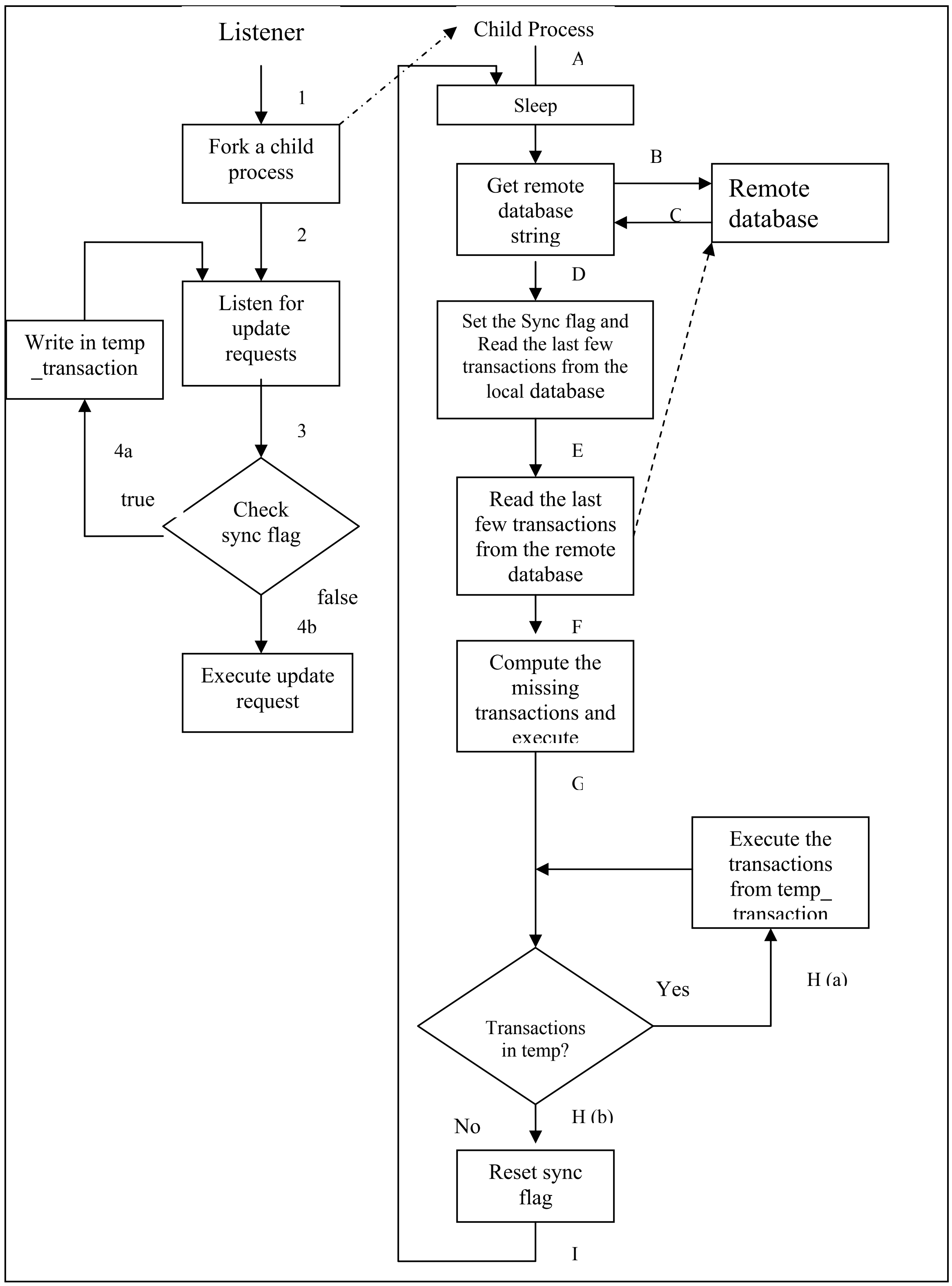
The set of transactions to be imported is determined using the transaction identifier of the last update transaction executed on the local database instance. The transaction identifier of the last update transaction on the local database instance is located in the Transaction table of the remote database instance. All the transactions entries from the remote database instance after that transaction identifier entry are imported to synchronize the local database instance. If the Transaction table on the local database instance is empty, the child process retrieves all the transactions from the Transaction table of the remote database instance. Once the new transactions have been retrieved from the remote database, the child process executes these new transactions in an order based on their timestamps [step E]. This ensures that the new transactions are executed in the same order as they were sent by the clients. Based on the number of new transactions that need to be executed, it might take a considerable amount of time to realize (execute) all the new transactions on the local instance of the database. During this interval, all the update requests received by the Listener are written in the Temp_transaction table as mentioned earlier.

After all the new transactions have been executed on the local database instance, the child process checks the Temp_transaction table to see if there are any update requests written to the Temp_transaction table [step F]. If there are any update requests in the Temp_transaction table, the child process tries to execute them [step G (a)]. If there are no update requests in the Temp_transaction table, the child process resets the Database_sync flag to 'false' indicating that the synchronization has completed [step G (b)]. The Listener upon receiving the next update request finds the Database_sync flag to be 'false' and executes the update request instead of writing it in the Temp_transaction table [step 5b]. After resetting the Database_sync flag the child process exits [step H]. At this point a new database instance or a recovering database instance is synchronized by the Listener with the other database instances on the network.

The above method of synchronizing a database is acceptable if the number of new transactions needed to be executed on the new database or recovering database are not many. If there are a huge number of new transactions needed to be executed, the above method may take a considerable amount of time. Also, the above method will not provide complete data consistency if the remote database has the updated data, but the transaction table of the remote database does not have all the transactions that have been performed up to the present time. To address these issues, an alternative method of copying entire tables of data from a remote database is also implemented to synchronize the database. Using this method, the local database instance copies the data of all the tables from the remote database. This method is efficient when there are a huge number of new transactions to be performed or the transaction table of the remote database does not have all the transactions. This method is not suitable when there are a few new transactions to be performed as it is unnecessary to transfer entire tables of data when only few transactions need to be executed. The earlier method turns out to be efficient when there are a few new transactions to be performed. The decision as to which method to use is left to the discretion of the administrator starting a database instance.

A database instance is synchronized only when the Listener is started or when the database instance is made available on the network. After the synchronization has taken place there is a possibility that the database instance may not see all the update requests. This maybe due to a number of reasons such as the physical network being broken between the database instance and the client sending the update request, the multicast message not able to reach the Listener as multicast is not a very reliable method of communication and others. Hence, it is required to refresh the local database instance periodically with other available database instances once the Listener is running and the database instance is made available on the network. The procedure followed to refresh a database instance is similar to the one adopted to synchronize a database instance. Figure 3.6 shows the steps executed by the Listener to refresh the local database instance periodically.

Besides forking a child process to synchronize the database instance when the Listener is started, it also forks a child process that refreshes the database instance periodically [step 1]. After forking the child processes, the Listener listens for the update requests on the multicast address [step 2]. The child process forked for doing Database Refresh sleeps for a specified interval of time [step A]. The child process does not start refreshing the database instance as soon as it is forked because there is another child process forked at around the same time for synchronizing the database instance. There is no need to refresh the database when the database synchronization is taking place. After sleeping for a specified interval, the child process resumes execution and sends a multicast message on the network to get the database string to connect to a remote database instance [Step B]. The other database instances on the network respond to the multicast message with a unicast message [step C]. The unicast message contains the database address to be used to connect to the remote database instance.



Listener
1
Fork a child process
2
Listen for update requests
Write in temp _transaction
3
4a
true
Check sync flag
false
4b
Execute update request
Child Process
A
Sleep
Get remote database string
B
C
Remote database
D
Set the Sync flag and Read the last few transactions from the local database
E
Read the last few transactions from the remote database
F
Compute the missing transactions and execute
G
Execute the transactions from temp_ transaction
Transactions in temp?
Yes
H (a)
No
H (b)
Reset sync flag
I

Figure 3.6: Database Refresh

After retrieving the remote database address, the child process starts the process to refresh the database by setting the Database_sync flag. The child process then retrieves the last few transactions observed by the local database instance within a specified time interval [step D]. The child process connects to the remote database instance and retrieves the transactions observed by the remote database instance within the same time interval [step E]. The child process now has two sets of transactions observed within the same time interval. One is the set of transactions from the local database instance and the other is the set of transactions from the remote local database instance for the same time interval. If the local database instance has missed any transactions, the local set of transactions will have lesser transactions than the remote set of transactions. The child process determines if there are any unrealized (missed) transactions and tries to execute them on the local database instance [step F]. Since the Database_sync flag is set by the child process, any update requests received by the Listener are written to the Temp_transaction table.

When the child process finishes executing the missing transactions, it reads the Temp_transaction table to see if any update requests have been written to it [step G]. If the Temp_transaction table has any update requests, the child process executes those requests on the local database instance [step H (a)]. If there are no update requests in the Temp_transaction table, the child process resets the Database_sync flag [step H (b)]. Resetting the Database_sync indicates that the database has been refreshed and the Listener can resume normal execution of executing update requests on the local database instance. The child process again sleeps for a specified time interval after which it tries to refresh the local database instance [step I]. The child process keeps refreshing the local database instance periodically as long as the Listener is running. Thus, the process of Database refresh along-with the process of Database synchronization ensures that the data in the local database instance is consistent with the data in the other available database instances. The next section discusses the fault manager that periodically checks the availability of the instances of a web service cluster.

3.4 Fault Manager (Monitor)

After a web service cluster has been started through the Web Service Manager, the availability of the instances in the cluster is periodically checked. Any instance on any server (machine) is prone to failures due to a number of reasons, such as the server (machine) shutting down abruptly or the web service instance encountering an unexpected exception. The periodic monitoring is done by a program, called Monitor, which checks the availability of all the instances in the cluster. The source code of Monitor can be found in Appendix B.

The Monitor program communicates with the web service instances running on different servers (machines) to determine their availability. The Monitor retrieves the access point(s) of the web service instances of a cluster from the Oracle database using the UUID assigned to the web service cluster when the cluster was started. It then calls the web service instances on different servers (machines) one at a time and inquires about the status of the instance.

The web service instances implement a generic function called getStatus() which is called by the Monitor to inquire about the status of the instance. The generic function sends back an “OK” response whenever it is called. A web service instance responding to the getStatus() function call is an indication that the web service instance is running. If the Monitor receives an “OK” response from the web service instance, no action is taken by the Monitor. If the Monitor is not able to communicate with the web service instance it is an indication that the web service is not available. The Monitor then retrieves the back-up server’s (machine) address from the Oracle database and tries to start a new instance on that server (machine) by communicating with the GenericSoapServer on the back-up server (machine). The Monitor modifies the information stored in the Oracle database for that web service cluster by deleting the access point of the instance that failed and adding the access point of the new instance that was started.

Since the Monitor is supposed to run periodically, it can be set to run as a cron-job on the server. A cron-job runs automatically at predefined time intervals on the server. Cron-jobs are used for carrying out repetitive tasks such as backing-up data, re-indexing a database or other tasks that require periodic execution. More information on how to set up a cron job can be found in the Unix Manual Pages [14].

The Web Service Manager communicates with the GenericSoapServer to start or stop a web service instance. The Monitor communicates with a web service instance to determine its availability. Thus, the Web Service Manager and the Monitor make calls to a web service. The Web Service Manager and the Monitor use the GenericSoapClient to make remote function calls to the GenericSoapServer and the web service instance respectively. The next section discusses GenericSoapServer and GenericSoapClient in detail.

3.5 GenericSoapServer and GenericSoapClient

The GenericSoapServer is deployed on the servers (machines) where web service instances are to be started. Each server (machine) has only one copy of GenericSoapServer running on it. It can be started as a part of the server's (machine) boot-up process. The GenericSoapServer is a web service built using GLUE. The source code of the GenericSoapServer can be found in Appendix B.

The GenericSoapServer is implemented as a web service and hence, the Web Service Manager should know the access point of the GenericSoapServer to communicate with it. Likewise the Monitor should know the access point of the web service instance it needs to communicate with to determine its availability. When a web service cluster is started, the Web Service Manager is given the access points of the GenericSoapServers to be contacted to start web service instances. The Web Service Manager is also given the access points of the GenericSoapServers running on the back-up servers (machines) so that these GenericSoapServers can be called to start a new instance in the event of failure of any running instance. These access points of the GenericSoapServers are stored in the Oracle database by the Web Service Manager.

The Web Service Manager retrieves the access points of the GenericSoapServers from the database whenever the Web Service Manger is instructed to start or stop a web service instance.

The GenericSoapServer implements two functions called startService() and stopService() to start a web service instance and to stop a web service instance respectively. These functions are called by the Web Service Manager to start and stop an instance respectively.

The GenericSoapServer uses the Gnu Database Manager (GDBM) on its local server (machine) to maintain information about the web service instances running on that server. Each copy of the GenericSoapServer running on different servers (machines) has its own local GDBM to store information about the web service instances running on that server. The interface provided by GDBM can be found on the Unix Manual pages [14]. For every web service instance started on a server (machine), the GenericSoapServer stores the following information in the GDBM:

a) Instance identifier – A UUID of the web service cluster to which the instance belongs. An instance running on any server (machine) for a given cluster is recognized using this identifier. It is used as a key to store information in the GDBM database.

b) Process identifier - The process number assigned by the system to the process executing a web service instance.

c) Access Point - The address or URL needed by the clients to access a web service instance.

The operations performed by the GenericSoapServer to start and stop a web service instance and by the Monitor to get the status of a web service instance are explained in the following paragraphs.

Let us suppose, an instance of the web service ‘simpleService’, described in Chapter 2, needs to be started by the Web Service Manager. The Web Service Manager calls the function startService() of the GenericSoapServer on the server (machine) where the instance is to be started. The parameters supplied by the Web Service Manager are the name of the web service (simpleService) and the unique identifier (UUID) (say ABCDEFGH) of the web service cluster to which the instance belongs.

The program (binary) that is to be executed to start the web service instance is placed on the server (machine) with a pathname known to the GenericSoapServer. The program (binary) is named after the web service. The GenericSoapServer finds the program (binary) using the web service name supplied by the Web Service Manager. If the GenericSoapServer is able to find the program (binary), it starts the web service instance as a new process on the system. The system allocates a process identifier (say 12345) to the web service instance of 'simpleService'. After successfully starting the instance, the GenericSoapServer enters the process number (12345) and the cluster identifier (ABCDEFGH) in the GDBM database. When the web service instance begins execution, it enters its access point (say http://beagle.rnet.missouri.edu:8009/soap/simpleService) in the GDBM database. Thus, the GDBM has three pieces of information stored for any web service instance started; the instance identifier, the process identifier and the access point. The GenericSoapServer reads the access point of the instance from the GDBM using the instance identifier and returns it to the Web Service Manager. The Web Service Manager in turn, enters the access point of the instance started in the Oracle database using multicast. Once an instance is started, it is available and locatable to provide its services to the clients. The following paragraph explains how a web service instance is stopped by the Web Service Manager.

Let us suppose, the Web Service Manager needs to stop an instance of the web service 'simpleService', described in chapter 2. It calls the GenericSoapServer on the server (machine) where the instance is to be stopped. The Web Service Manager calls the stopService() function and passes along the instance identifier (ABCDEFGH). Upon receiving the request, the GenericSoapServer looks up the GDBM to find an entry that matches the instance identifier (ABCDEFGH). If an entry exists, the GenericSoapServer retrieves the process identifier (12345) from the GDBM and terminates the process using the system call kill(). The system call kill() is used on a UNIX system to terminate a process. The kill() system call needs to be passed the process identifier to kill the process. The usage of the kill() system call can be found in the Unix Manual Pages [14]. After the process (instance) is terminated, the GenericSoapServer removes the entry of the instance from the GDBM. The GenericSoapServer returns a message to the Web Service Manager indicating whether or not the instance was successfully terminated. If the instance is successfully terminated, the Web Service Manager removes the information for that instance from the Oracle database using multicast.

The Monitor periodically retrieves the access points of the web service instances from the Oracle database to perform the availability check of the instances. If the instance has failed, the Monitor starts a new instance on the back-up server (machine) and updates the access point in the Oracle database using Multicast.

The GenericSoapServer is a web service written in Java. The GenericSoapServer is called by the Web Service Manager. Also the Monitor (Fault Manager) communicates with web service instances to check their availability. To make remote function calls to the GenericSoapServer and the web service instances by the Web Service Manager and the Monitor respectively, a web service client is written in Java called GenericSoapClient. It implements functions that can be called by programs (Web Service Manger and Monitor) that want to communicate with a web service. The source code of GenericSoapClient can be found in Appendix B. The block diagram of the design with the introduction of GenericSoapClient is shown in Figure 3.7.

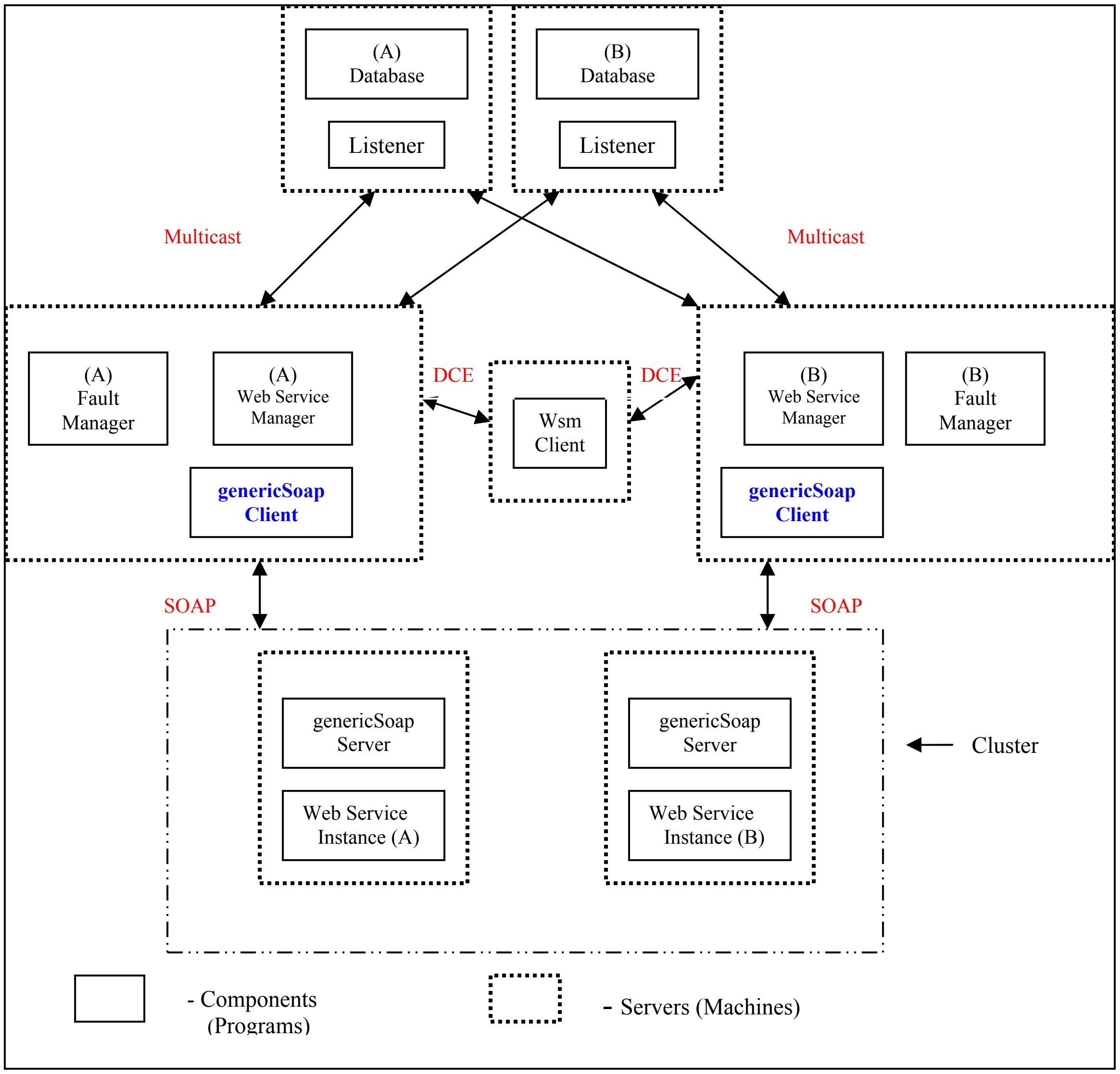


Figure 3.7: The Block Diagram of the Design with the Introduction of GenericSoapClient

The Web Service Manager and the Monitor are implemented in the C Language. The GenericSoapClient is written in Java and thus, a call to a Java function (GenericSoapClient) is required to be made from a C program (the Web Service Manager or the Monitor). This is accomplished with the help of the Java Native Interface (JNI), an interface that contains the necessary libraries to enable JAVA to interoperate with C and vice-versa. A reference to use the JNI can be found in "Java 2: The Complete Reference" [12]. The following paragraph describes how a Web Service Manager or a Monitor uses GenericSoapClient to call a web service.

Let us suppose, the Monitor needs to call the getStatus() function of the web service instance to check its availability. The Monitor passes the access point of the web service instance (say http://beagle.rnet.missouri.edu:8009/soap/simpleService) to be contacted to the GenericSoapClient. The Monitor also passes the name of the function (getStatus) to be called and the input parameters to the function if any, to the GenericSoapClient. The GenericSoapClient calls the getStatus() function of the web service instance at http://beagle.rnet.missouri.edu:8009/soap/simpleService and passes the input parameters if any to the function. After the call is successfully completed, the GenericSoapClient returns the results obtained from the call to the Monitor. This shows how the Monitor and a web service instance communicate using the GenericSoapClient. Similarly, the Web Service Manager calls the startService() and the stopService() functions of the GenericSoapServer using GenericSoapClient.

The Web Service Manager is provided with a web service interface so that the instances of a web service cluster can communicate with the Web Service Manager using SOAP. The next section discusses the need and implementation of a web service interface of the Web Service Manager.

3.6 WsmSoapServer (Web Service interface of the Web Service Manager)

As explained in Section 3.4, a failure of a web service instance is detected by the Monitor, but the detection may not be at the same moment the instance fails. Since Monitor is run periodically and there are many instances in a web service cluster that the Monitor has to check, there could be some time lag before the Monitor detects a failed instance. Hence, if possible, it is desirable to have the web service instance send a notification of failure to the Web Service Manager as soon as it fails. Depending upon the nature of failure, the instance may or may not be able to notify the

Web Service Manager about the failure, in which case, the Monitor detects the failure sometime later.

The Web Service Manager is implemented as a DCE server and hence, an instance that needs to communicate its failure to the Web Service Manager should have the necessary client stub and the RPC runtime library installed on the server (machine) to communicate with the Web Service Manager. To avoid this requirement of installing the client stub and the RPC runtime library on the servers (machines) that run the web service instances, a web service interface is provided to the Web Service Manager. The web service interface of the WSM is a web service called WsmSoapServer written in JAVA.

The SF toolkit needed to write a client program to call the WsmSoapServer is easier to install and maintain on a server (machine). The web service instances communicate the failure to WsmSoapServer using SOAP. The WsmSoapServer in turn, forwards the notification of a failure to the Web Service Manager using DCE. Thus, the WsmSoapServer acts as an intermediary between the web service instances and the Web Service Manager. The WsmSoapServer is started on servers (machines) that have the necessary client stub and the RPC runtime library to communicate with a DCE server. Since WsmSoapServer is a web service, it is started and administered as a cluster like other web service clusters. The block diagram of the design with the introduction of the WsmSoapServer is shown in Figure 3.8. The figure also shows an additional program Ws_admin, which is discussed in the next section. Figure 3.8 shows all the components of the system.

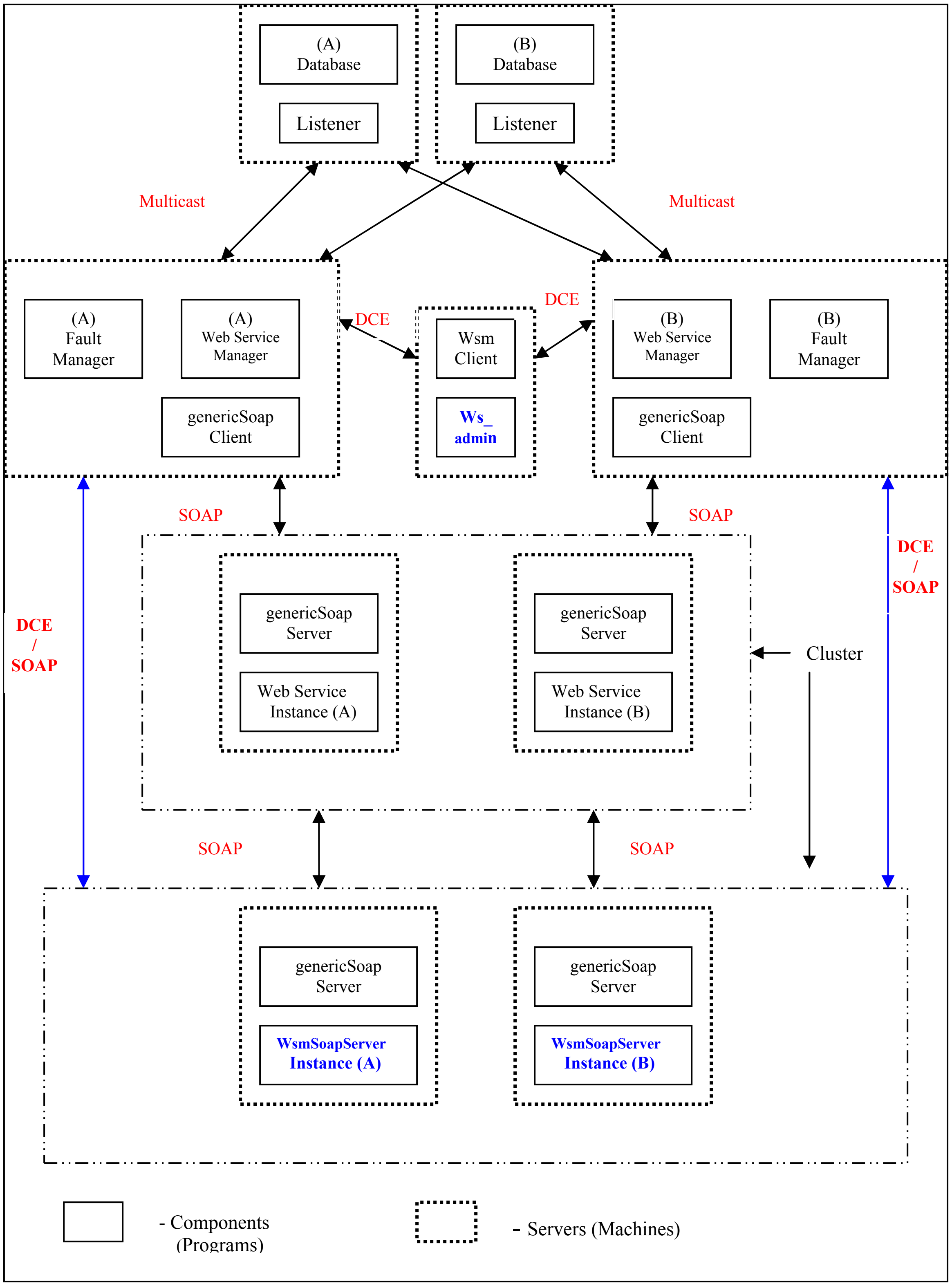


Figure 3.8: The Block Diagram of the Design with all the Components

The WsmSoapServer implements the function notify(), which is called by the web service instances to notify the Web Service Manager of their failure. The web service instance(s) that needs to communicate with WsmSoapServer gets the access point of the WsmSoapServer by contacting the Web Service Registry. The Monitor periodically checks the availability of the WsmSoapServer instances as it does for other web service clusters. The notification of its own failure is communicated by the WsmSoapServer to the Web Service Manager using DCE. The source code of the WsmSoapServer can be found in Appendix B.

The Web Service Manager implements the procedures (functions) to start and administer a web service cluster. It also implements procedures to manage administrative accounts. Since the Web Service Manager is a DCE server, an interface (program) is needed to call these procedures remotely. This is achieved with the help of DCE client programs called Wsmclient and Ws_admin that interact with the Web Service Manager. The following section discusses the DCE client programs, Wsmclient and Ws_admin.

3.7 Wsmclient and Ws_admin

The Wsmclient is a DCE client program that implements a menu-driven interface to accept input from an administrator and calls the appropriate remote procedures of the Web Service Manager to start and administer a web service cluster. An administrator is a person who administers a web service cluster. When executed, the Wsmclient contacts the DCE CDS to get the address(s) of all the host machines running the Web Service Manager. If the Wsmclient is not able to find any Web Service Manager to communicate with, it aborts and reports failure. If it finds any, it communicates with any one of the available Web Service Managers. The selection of the Web Service Manager to communicate with can be based on a number of criteria. For this thesis, the administrator is provided the list of the host servers (machines) running the Web Service Manager. The selection of the server (machine) is left to the discretion of the administrator running Wsmclient.

The Wsmclient is run by an administrator to administer a web service cluster. The usage of the Wsmclient is described in detail in Chapter 4. An administrator can run and manage more than one web service clusters. The Wsmclient can also be run by more than one administrator, each administrating their own cluster(s) of web service(s). Hence, it is important to verify the identity of each administrator before he is given full access to administer a web service cluster. Also, it is imperative to see that the administrator can only administer the web service cluster owned by him. For this purpose, the administrator needs to be authenticated to access the Web Service Manager and authorized to administer only the web service cluster(s) owned by him. The form of authentication used is the basic username-password mechanism. The password is only known to the administrator and hence by providing the correct password, the administrator authorizes himself to administer the web service cluster(s) owned by him. Thus, an administrator's account is required to be created. This is done by the super-administrator.

The role of the super-administrator is to create an administrator's account to be used to run Wsmclient. Thus, the super-administrator keeps control of the number of administrator accounts created to allow access to Wsmclient. The information maintained in the Oracle database for each administrator account includes the username, the password and optional information such as the address, the email and the phone number.

Once an administrator's account is created, the administrator can run Wsmclient to start and administer a web service cluster. Each administrator's account when created is assigned a unique identifier (UUID). Each web service cluster started by the Web Service Manager is associated with this UUID of the administrator. Whenever the administrator tries to administer a web service cluster, an authorization check is made to see if he owns the web service cluster using the UUID and, if the cluster he wants to administer is not owned by him an error message is returned to the administrator.

The Web Service Manager also implements the procedures to be called to create and mange administrative accounts. These procedures are remotely called by an administrative program called Ws_admin. Ws_admin is executed only by the super-administrator. This program is discussed below.

The Ws_admin is a DCE client program that implements a menu-driven interface to accept input from a super-administrator. Based on the input entered by the super-administrator, it calls the appropriate remote procedures of the Web Service Manager to create and manage administrative accounts. Like Wsmclient, Ws_admin also contacts the DCE CDS to get the address(s) of all the servers (hosts) running the Web Service Manager and displays the list of the servers (machines) to the super-administrator. The selection of the server to access is left to the discretion of the super-administrator.

The functions that can be performed using Ws_admin include creating an account, deleting an account, resetting an account and listing the accounts that have been created. The usage of Ws_admin is described in detail in Chapter 4. There are two types of accounts that can be created and managed using Ws_admin. One is the administrator's account needed to run Wsmclient (WSM account) and other is the user's account needed to access the Web Service Registry (WSR account). The need to have a WSR account to access the Web Service Registry is discussed in Chapter 5.

After having introduced all the components of the fault-tolerant system, the next chapter (Chapter 4) discusses the usage of the fault-tolerant system through Wsmclient and Ws_admin. It discusses the procedures executed to create and manage an administrator's (or user's) account, and also to start and administer a web service cluster. It discusses the interaction among the various components to execute these procedures and the data exchanged among the components.

4. Usage of the Fault-Tolerant System to Administer Fault-Tolerant Web Services

This chapter discusses the usage of the fault-tolerant system, discussed in Chapter 3, in terms of the procedures executed on the fault-tolerant system to start and administer a web service cluster. The procedures discussed perform the administrative tasks of starting a web service cluster, removing a web service cluster, creating an administrator's account, and other tasks. This chapter discusses the steps taken by an administrator to execute a procedure on the system, the input data entered by the administrator and the interaction among the components of the system. This chapter acts as a guide to an administrator; ranging from deploying the various components of the fault-tolerant system to understanding the various procedures to start and administer a fault tolerant web service cluster.

Before the procedures are discussed, the following section lists the components to be deployed to start the fault-tolerant system. It also specifies, for some components, the requirements, software or the resources needed to execute and interact with the other components of the system.

4.1 Deployment of the Components

The Listener is the first component of the fault-tolerant system to be started. It is deployed on servers (machines) that have an Oracle database instance running. At-least two instances of the Oracle database must be available on the network to avoid a single point of failure. The fault-tolerant system will run even with only one instance of the Oracle database, but then the system is prone to a single point of failure. The Listener is contacted by other components of the system to get the address of the local Oracle database instance and to update the database instance. The other components of the system communicate with the Listener using multicast. For the Listener to receive multicast messages, the network where the Listener is deployed must support multicast.

The schema (database tables) used to store information in the Oracle database and the script to create the schema on the database instance is available in Appendix A.

The Web Service Manager is a DCE server which implements the procedures to start and administer a web service cluster by an administrator. It also implements procedures to create accounts for the administrators so that the administrators can run Wsmclient and have access to the fault-tolerant system. The Web Service Manager is deployed on the servers (machines) that support the execution of a DCE server. The number of instances of the Web Service Manager started is left to the discretion of the administrator. At-least two instances of the Web Service Manager must be started to avoid a single point of failure. The fault-tolerant system will run even with only one instance of the Web Service Manager, but then the system is again prone to a single point of failure. The server (machine) running the Web Service Manager must have an Oracle client installed. The Oracle client is used to establish a connection to a remote Oracle database instance. The Oracle client is needed by the Web Service Manager to do a remote connection to the available Oracle database instance(s) to perform a read operation. The network where the Web Service Manager is deployed must support multicast to enable the Web Service Manager to send out the update multicast messages to the Listener(s).

The Web Service Manager communicates with the GenericSoapServer on servers (machines) where the web service instances are to be administered. Since the GenericSoapServer is called using the GenericSoapClient, the GenericSoapClient program is also required to be deployed on the same server (machine) as the Web Service Manager. When the Web Service Manager is started, it contacts the Listener(s) to get the address(s) of the database instance(s) using multicast. The Web Service Manager uses this address to remotely connect to a database instance to do a read operation.

The Monitor runs periodically to check the availability of the web service instances of all web service clusters and hence, it can be started as a cron-job, set to run at regular intervals. It can be deployed on the same server (machine) as the Web Service Manager, as it needs the Oracle client to do a read operation. The Monitor also requires the GenericSoapClient to communicate with the web service instances of a web service cluster in order to check their availability. The network where the Monitor is running should support multicast. Like the Web Service Manager, the Monitor contacts the Listener(s) to get the address(s) of the database instance(s) using multicast.

The GenericSoapServer receives calls from the Web Service Manager to start and stop a web service instance. The GenericSoapServer is started on the servers (machines), where the web service instances are to be started and administered. The binary (executable), needed to start a web service instance is placed on the same server (machine) with a pathname known to the GenericSoapServer. The GenericSoapServer finds the binary (executable) of the web service and executes the binary as a separate system process.

The Wsmclient and Ws_admin programs are deployed on a server (machine) that has the RPC runtime library installed to make remote procedure calls to a DCE server. The Wsmclient and the Ws_admin programs, based on the task selected, call the appropriate remote procedures of the Web Service Manager. The Wsmclient is used by an administrator to administer a web service cluster. The Ws_admin is used by a super-administrator to create accounts for the administrators to provide them access to Wsmclient. The super-administrator is a person who has the authority to create administrator accounts.

Once the components are deployed on the respective servers (machines), the fault-tolerant system can be accessed using the programs Wsmclient and Ws_admin. The super-administrator, when running the Ws_admin, is provided with an interface to execute various procedures to create and manage an administrator’s account. Once an administrator’s account is created, the administrator can access the fault-tolerant system using the Wsmclient program. The Wsmclient program provides an interface to execute the various procedures to start and administer a web service cluster. The following section gives a generalized view of the interaction among the various components of the fault-tolerant system to accomplish different tasks using Wsmclient and Ws_admin.

4.2 Generalized View of the Interaction Among the Components of the Fault-Tolerant System

Figure 4.1 shows the generalized steps and the interaction among the components to execute any given task (procedure) on the fault-tolerant system. When the Ws_admin program or the Wsmclient program is started by the super-administrator or the administrator respectively, the program contact the DCE CDS to get the hostnames of the servers (machines), where the Web Service Managers are running [step 1]. The DCE CDS responds with a set of hostnames of the servers (machines) that run the Web Service Manager as a DCE server [step 2]. The super-administrator or the administrator is provided the set of hostnames of the servers (machines) running the Web Service Managers. The super-administrator or the administrator selects the hostname of a server (machine) from the list of the available servers (machines) to communicate with the Web Service Manager [step 3].

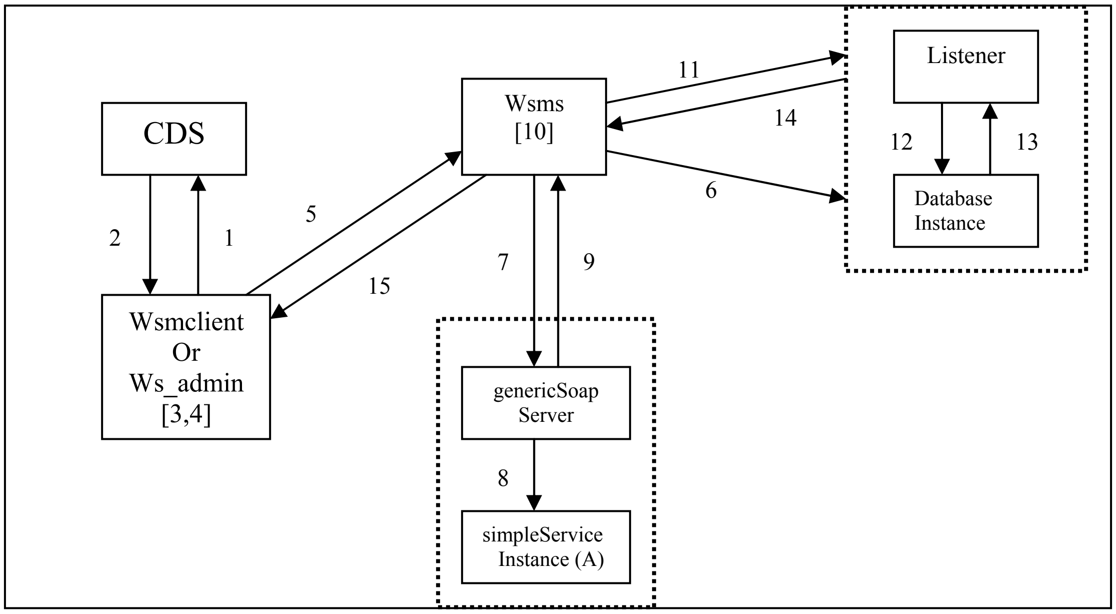


Figure 4.1: Generalized View of the Interaction Among the Components of the System

The super-administrator or the administrator selects the task to be executed and enters the information required to execute the task (procedure) [step 4]. After the information has been entered by the super-administrator or the administrator, the Ws_admin or the Wsmclient calls the appropriate remote procedure of the Web Service Manager on the selected server and passes the information collected from the super-administrator or the administrator [step 5]. The steps executed by the Web Service Manager vary depending upon the task (procedure) being executed.

If the remote procedure of the Web Service Manger called is to start or administer a web service cluster by the Wsmclient, the Web Service Manager does a check to authenticate the administrator by doing a read operation from the database instance [step 6]. If the administrator fails the authentication check, a message is sent back to Wsmclient indicating that the administrator has not been authenticated. If the authentication is successful, the Web Service Manager communicates with the GenericSoapServer to either start or stop a web service instance, based upon the task selected by the administrator [step 7].

The GenericSoapServer starts or stops the web service instance [step 8] and returns an appropriate message back to the Web Service Manager indicating whether or not the operation was successful [step 9]. The Web Service Manager prepares a Structured Query Language (SQL) [18] to update the database instance [step 10]. SQL is a language used to read and update information in an Oracle database instance. The SQL query is specific to the task executed on the system. If the task selected is to create a new account for an administrator using the Ws_admin, the Web Service Manger prepares a query to insert information about the new account in the database. If the task selected is to start a web service instance using the Wsmclient, the Web Service Manager prepares a query based upon the response received from the GenericSoapServer, to update the information in the database instance.

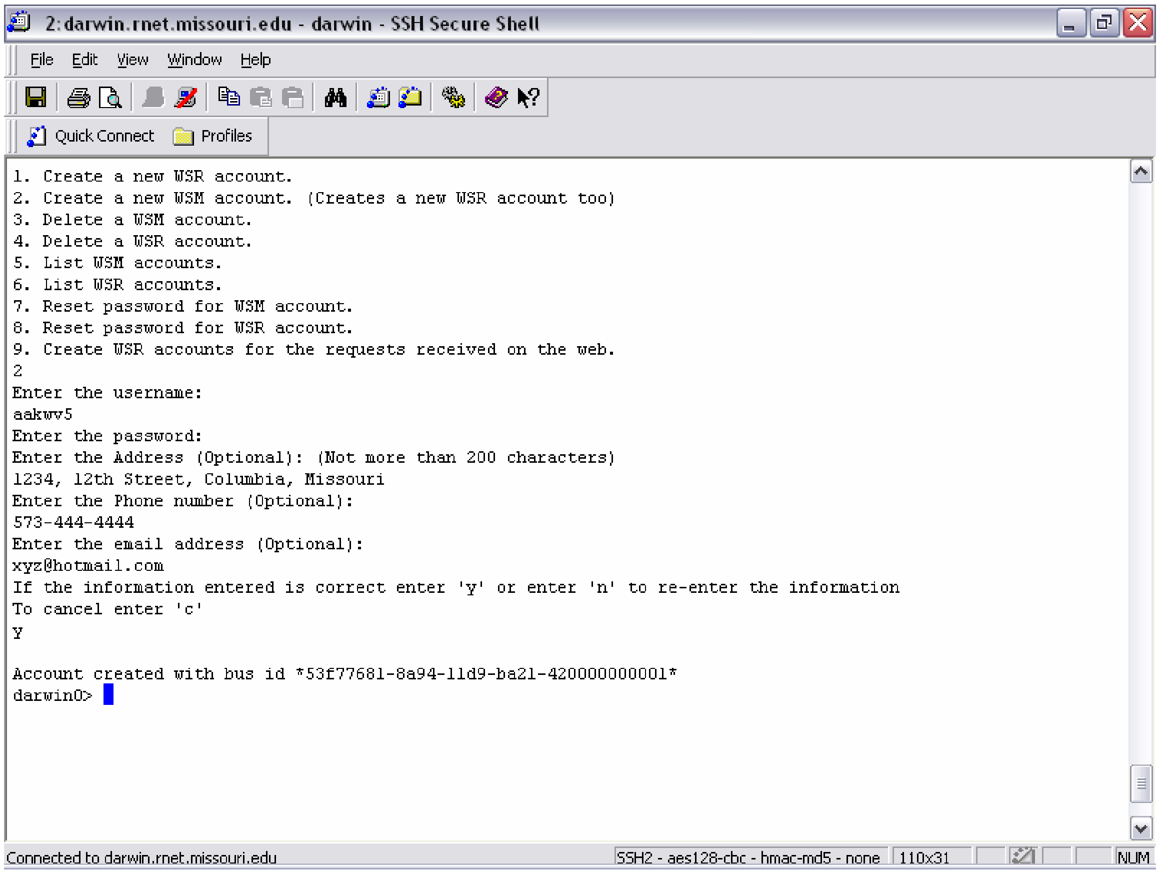
After the SQL query is prepared, the Web Service Manager packs the SQL query in an update request message and sends it on the multicast address to the Listener(s) [step 11]. The update multicast message also carries a unique transaction identifier. The Listener(s) receives the multicast message, extracts the query from the update request message and executes the request on the local database instance [step 12]. The Listener(s) logs the SQL query along-with the transaction identifier in the log maintained in the Transaction table in the database. The database instance updates the data and returns an appropriate response back to the Listener(s) [step 13]. The result of the execution of the update request is sent back to the Web Service Manager by the Listener(s) using unicast [step 14]. The Listener(s) sends an "OK" response if the update request was executed successfully or sends an error message. The Web Service Manager, upon receiving a response from the Listener(s), sends a message back to the super-administrator or the administrator, indicating whether or not the execution of the task was successful [step 15]. If the remote procedures of the Web Service Manager called are to manage administrative accounts by the Ws_admin, the Web Service Manager reads or updates the Oracle database instance.

After having understood the generalized interaction among the components of the fault-tolerant system to execute various tasks, the following sections give a detail explanation of each task (procedure) that can be executed on the fault-tolerant system. The explanation includes the data entered by the super-administrator or the administrator to execute each task (procedure), the steps taken by the Web Service Manger specific to each task (procedure) and the data that gets updated in the database instance(s).

An administrator who wants to start and administer a web service cluster needs to have an administrator's account to be authenticated and authorized to use the Wsmclient. This is done to ensure only specific users can execute the Wsmclient. The creation of an account includes assigning a username and password to the administrator. The following section describes the procedure followed by the super-administrator to create and manage an administrator's account using Ws_admin.

4.3 Create and Manage an Administrator Account Needed to Run Wsmclient

The Ws_admin program provides the super-administrator with a set of options to create and manage an administrator's account. This is shown in Figure 4.2. The options allow the super-administrator to create, delete, reset and list the administrator (WSM) accounts. In addition to administering the administrator accounts needed to run the Wsmclient program, the super-administrator also uses Ws_admin to create and manage a user account called a Web service registry (WSR) account needed to access the web registry. This section discusses the administrator account needed to run the Wsmclient program. The WSR accounts are discussed in Chapter 5, which discusses the Web Service Registry in detail.


2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
110x31
NUM

```
1. Create a new WSR account.
2. Create a new WSM account. (Creates a new WSR account too)
3. Delete a WSM account.
4. Delete a WSR account.
5. List WSM accounts.
6. List WSR accounts.
7. Reset password for WSM account.
8. Reset password for WSR account.
9. Create WSR accounts for the requests received on the web.
2
Enter the username:
aakwv5
Enter the password:
Enter the Address (Optional): (Not more than 200 characters)
1234, 12th Street, Columbia, Missouri
Enter the Phone number (Optional):
573-444-4444
Enter the email address (Optional):
xyz@hotmail.com
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

Account created with bus id *53f77681-8a94-11d9-ba21-420000000001*
darwin0>
```

Figure 4.2: Options Offered by Ws_admin and the Steps Followed to Create a WSM Account

The creation of an administrator account also results in the creation of a WSR account, but not vice-versa. Starting a web service cluster using Wsmclient automatically publishes the web service in the Web Service Registry so that the web service is advertised to the clients. The Web Service Registry is a tool used by the developers to publish their web services and by the clients to search for those web services based on some search criteria. A WSR account is required to publish a web service in the Web Service Registry. Hence, a WSR account is also created at the creation of an administrator's account with the same username.

An administrator account is created by following the second option of the Ws_admin as shown in Figure 4.2. The super-administrator is prompted to enter the information needed to create an administrator account. The information includes the administrator's username and a password along-with some optional information such as the address, email and telephone number of the administrator. The optional information is useful to display information about the owner of the web service by the Web Service Registry.

After the required information has been entered by the administrator, the Ws_admin calls the appropriate remote procedure of the Web Service Manager and passes the information collected from the super-administrator. The password is encrypted before it is sent across the network. Sine a new account is to be created, a universally unique identifier needs to be generated for the new account. The Web Service Manager generates a universally unique identifier (UUID) for the new account by using the current timestamp, the identifier of the server (machine) running the Web Service Manager and the encrypted string of the administrator's username. These three parameters ensure that the identifier generated for the account is universally unique. After the UUID is generated, the Web Service Manager prepares a SQL query to insert a record for the new account in the database and sends it in an update message on the multicast address to the Listener(s). The Web Service Manager, upon receiving a response from the Listener(s), sends a message back to the super-administrator, indicating whether or not the account was successfully created. If an account already exists with the same username, the account is not created and the super-administrator is requested to provide a different username. Upon successful creation of an account, the super-administrator is displayed the UUID of the administrator account created called Business Identifier (busid). This is shown in Figure 4.2.

Once the administrator's account has been created, the administrator is identified by the UUID assigned to him and the administrator can run Wsmclient to start and administer a web service cluster. The administrator has to authenticate himself by providing the username and the password each time he executes the Wsmclient program. An administrator can administer more than one web service cluster. Since an administrator can administer more than one web service cluster, an authorization check is also done in addition to authenticating the administrator to see if the administrator is the owner of the web service cluster.

As mentioned earlier, Ws_admin also provides the super-administrator the capability to delete, reset and list the administrator accounts in addition to creating an administrator account. The third option shown in Figure 4.2 prompts the super-administrator to enter the username of the account to be deleted. After the information is entered, the Ws_admin calls the appropriate remote procedure of the Web Service Manager to delete the administrator account and passes it the username of the account to be deleted. The Web Service Manager sends an update request message on the multicast address to the Listener(s). The Listener(s) execute the query and send a response back to the Web Service Manager. The Web Service Manager in turn, sends a response back to Ws_admin indicating whether or not the WSM account was deleted. Similarly, following the fifth and the seventh option shown in Figure 4.2, all the administrator accounts can be listed or the password of an administrator account can be reset respectively.

To start a web service cluster, the administrator has to start a GenericSoapServer on the servers (machines) where the web service instances are to be started. The GenericSoapServer can be started as a part of the server's (machine) boot-up process, so that the GenericSoapServer is automatically started when the server (machine) is started. Once the GenericSoapServer is started on the servers (machines), the administrator can start a web service cluster through the Web Service Manager.

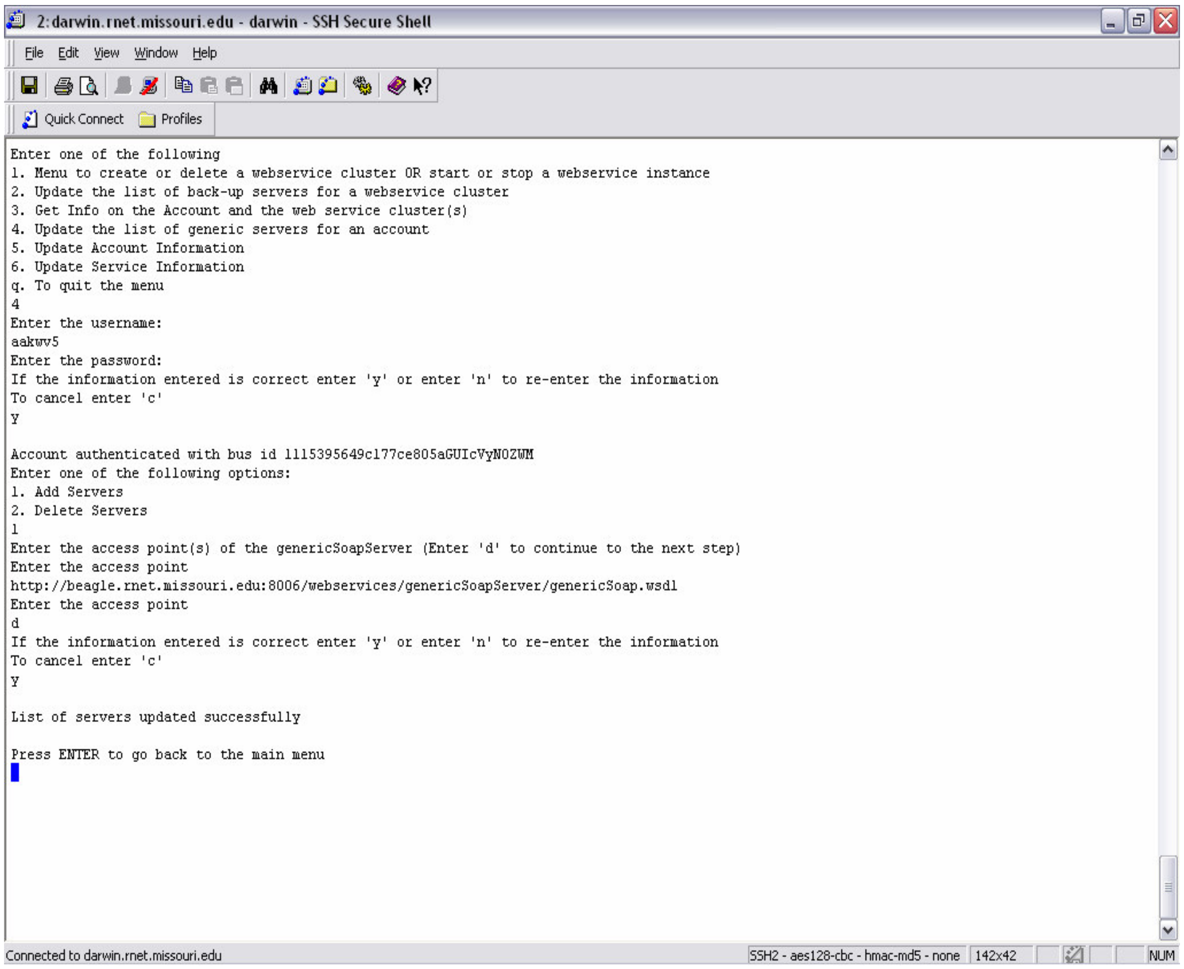
The GenericSoapServers are remotely contacted by the Web Service Manager to start and stop the web service instances. As the GenericSoapServer is implemented as a web service, the Web Service Manager needs to know the access points of the GenericSoapServers to start or stop web service instances. The administrator provides this information about the access points of the GenericSoapServers with other information when executing Wsmclient.

The access points of the GenericSoapServers to be entered are lengthy and hence, an incorrect access point can be entered by the administrator while running a procedure to administer a web service cluster. To reduce the possibility of it happening, the access points of the GenericSoapServers are stored in the Oracle database under the administrator's account. The administrator is required to carefully enter the access point(s) of the GenericSoapServers only once to be stored in the Oracle database. A set of access points are stored for each administrator. Whenever the administrator needs to enter the access point(s) of GenericSoapServers, the list of the access points of the GenericSoapServers is fetched from the database and displayed to the administrator. The administrator can select the access point(s) of the GenericSoapServers needed to start and manage a web service cluster from the list displayed to him.

Wsmclient prompts the administrator to enter the access points of the GenericSoapServers to be used when the administrator executes Wsmclient for the first time. Once the list of the access points of the GenericSoapServers has been created, the administrator can update the list of the access points. This way the administrator has to type in the access points only once when storing them in the database rather than typing the access points every time when needed. Using Wsmclient, the administrator can update the list of the GenericSoapServers. The following section describes the procedure to update the list of GenericSoapServers for a given administrator account.

4.4 Update GenericSoapServer List

The administrator can create the list, add to the list or delete from the list the access points of the GenericSoapServers using the fourth option of Wsmclient as shown in Figure 4.3. When the administrator executes the Wsmclient for the first time, the list of the access points of the GenericSoapServers is empty. The administrator has to enter the access points of the GenericSoapServers to create a list of the access points to be used to start and administer a web service cluster. Once the list is created, the administrator can update the list as and when required.


2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
142x42
NUM

```
Enter one of the following
1. Menu to create or delete a webservice cluster OR start or stop a webservice instance
2. Update the list of back-up servers for a webservice cluster
3. Get Info on the Account and the web service cluster(s)
4. Update the list of generic servers for an account
5. Update Account Information
6. Update Service Information
q. To quit the menu
4
Enter the username:
aakwv5
Enter the password:
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

Account authenticated with bus id 1115395649c177ce805aGUIcVyN0ZWM
Enter one of the following options:
1. Add Servers
2. Delete Servers
1
Enter the access point(s) of the genericSoapServer (Enter 'd' to continue to the next step)
Enter the access point
http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
Enter the access point
d
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

List of servers updated successfully

Press ENTER to go back to the main menu
```

Figure 4.3: Information Entered to Update GenericSoapServer List

After selecting the fourth option, the administrator is asked to enter the username and the password. The username and the password entered by the administrator must be the ones used by the super-administrator to create this administrator account. The password entered is encrypted and a remote procedure call is made to the Web Service Manager to authenticate the administrator passing the username and the password. The Web Service Manager remotely connects to the database instance and performs a read operation to verify the username and the password.

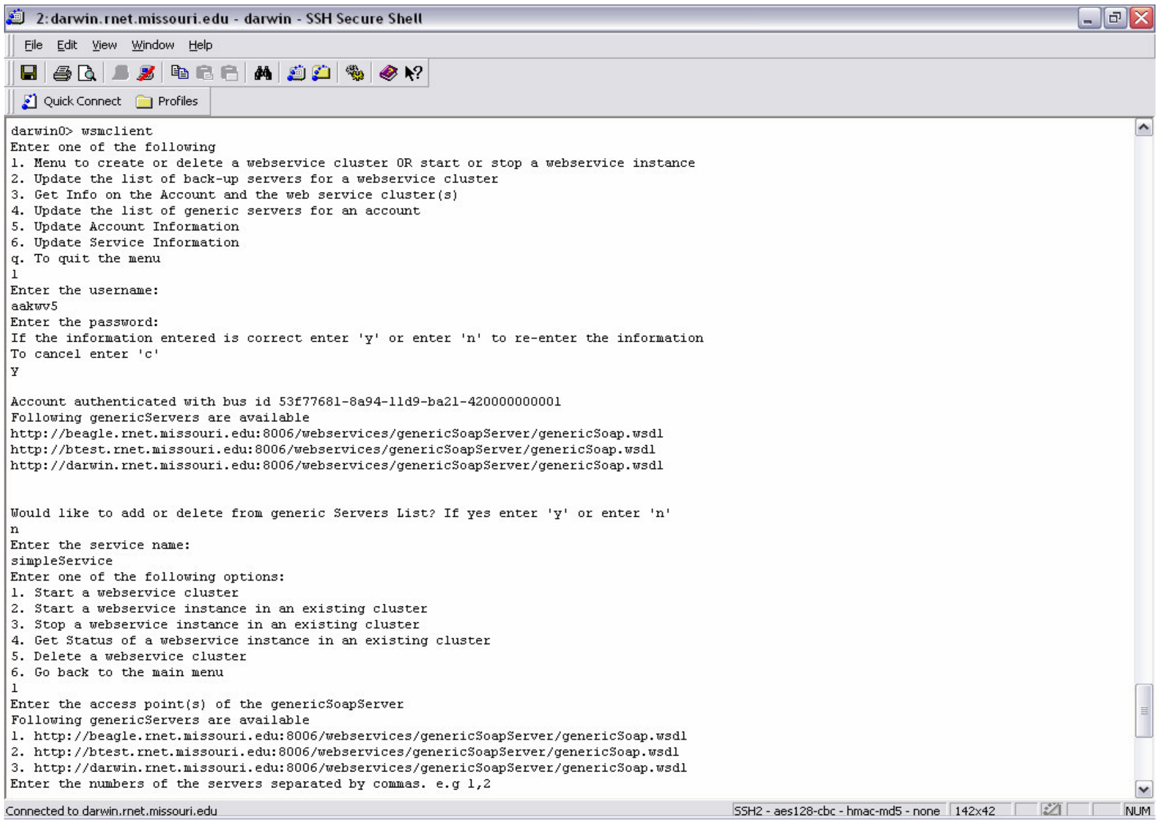
After the read operation, the Web Service Manager sends a response back to Wsmclient indicating whether or not the administrator is authenticated to use Wsmclient. If the administrator is authenticated, a set of options are shown to either add or delete an access point of the GenericSoapServers from the list, as shown in Figure 4.3. If the administrator is executing the Wsmclient for the first time, the list is empty and the administrator chooses the option to add access point(s) of the GenericSoapServers to the list. The administrator has to type in the access point(s) of the GenericSoapServers. The administrator has to be careful not to type in the incorrect access point of the GenericSoapServer. The administrator has to take this precaution only when adding access point(s) to the list. Once the access points are stored in the database, the administrator will be displayed the list of the access points to select from whenever the administrator is required to enter the access point(s) of the GenericSoapServers. The information entered by the administrator is shown in Figure 4.3. After the information is collected by the Wsmclient, it forwards the request to the Web Service Manager. The Web Service Manager sends a multicast update request to the Listener(s). The Web Service Manager notifies the administrator with an appropriate message indicating whether or not the update was successful.

To start a web service cluster the administrator must have a valid administrator's account, must deploy the GenericSoapServers on servers (machines) where the web service instances are to be started, must have the binary of the web services placed in a path known to the GenericSoapServer and must create a list of the access points of the GenericSoapServers in the database. If these requirements are met, the administrator can start a web service cluster. The following section describes the procedure adopted by the administrator to start a web service cluster.

4.5 Start a Web Service Cluster

The information needed by the Web Service Manager to start a web service cluster is the name of the web service, the access points of the GenericSoapServers running on the servers (machines), where the web service instances are to be started and, the access points of the GenericSoapServers running on the back-up servers (machines) to be used to start a new instance in the event of a failure of any of the running instances. Optional information such as the web service name to be used by the Web Service Registry, the URL of the documentation page, if any, and other information can also be supplied to the Web Service Manager while starting a web service cluster.

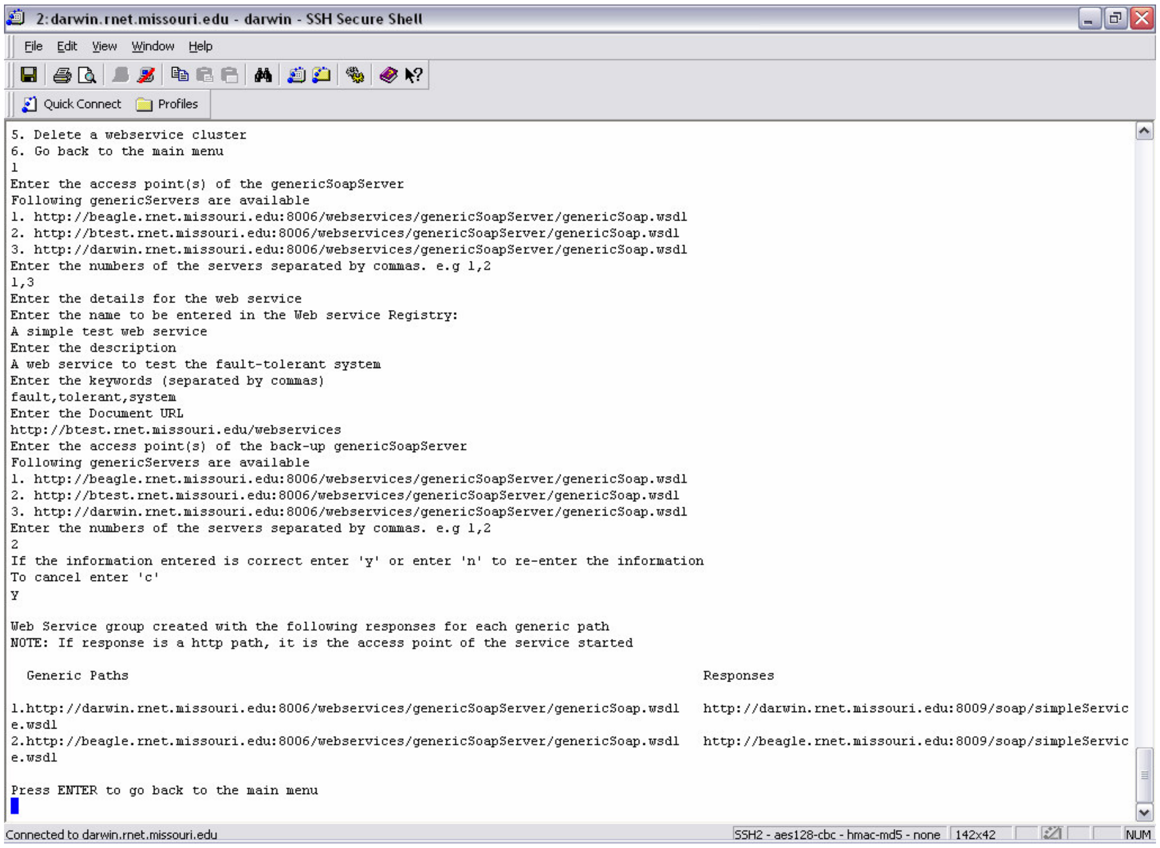
As shown if Figure 4.4a, the administrator selects the first option to start a web service cluster. The administrator is asked for the username and password. After the administrator is authenticated, the administrator is asked if he wants to update the list of the access points of the GenericSoapServers. If the administrator chooses to update the list of the access points of the GenericSoapServers, then the Wsmclient proceeds as explained in Section 4.4 and the list is updated. Whether or not the administrator chooses to update the list, the administrator is shown a new set of options to administer a web service cluster. This is shown in Figure 4.4a. Before the new options are shown, the administrator is prompted to enter the web service name to be used by the GenericSoapServer to locate the binary of the web service on the local server (machine) and execute it. The new set of options allows the administrator to create and delete a web service cluster; and to start, stop and determine the status of a web service instance. The first option in the new set of options allows the administrator to start a web service cluster. The other options listed are discussed in the following sections.


2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
142x42
NUM

```
darwin0> wsmclient
Enter one of the following
1. Menu to create or delete a webservice cluster OR start or stop a webservice instance
2. Update the list of back-up servers for a webservice cluster
3. Get Info on the Account and the web service cluster(s)
4. Update the list of generic servers for an account
5. Update Account Information
6. Update Service Information
q. To quit the menu
1
Enter the username:
aakwv5
Enter the password:
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

Account authenticated with bus id 53f77681-8a94-11d9-ba21-420000000001
Following genericServers are available
http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
http://btest.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl


Would like to add or delete from generic Servers List? If yes enter 'y' or enter 'n'
n
Enter the service name:
simpleService
Enter one of the following options:
1. Start a webservice cluster
2. Start a webservice instance in an existing cluster
3. Stop a webservice instance in an existing cluster
4. Get Status of a webservice instance in an existing cluster
5. Delete a webservice cluster
6. Go back to the main menu
1
Enter the access point(s) of the genericSoapServer
Following genericServers are available
1. http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
2. http://btest.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
3. http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
Enter the numbers of the servers separated by commas. e.g 1,2
```

Figure 4.4a: Options Offered by Wsmclient to Administer a Web Service Cluster

Figure 4.4b shows the prompts and the information entered by the administrator to start a web service cluster of the ‘simpleService’ web service. SimpleService is a generic web service that was described in Section 2.3. The administrator is shown the list of the access points of the GenericSoapServers to select the ones needing to be contacted to start the web service instances. The administrator enters the numbers corresponding to the access points in the list rather than entering the entire access points. After the access points of the GenericSoapServers have been selected, the administrator is prompted to enter the name, keywords and the description of the web service to be used by the Web Service Registry to display information about the web service. The administrator is also prompted to select GenericSoapServers needing to be contacted to start a web service instance on the back-up servers (machines). After all the information is entered, the Wsmclient calls the Web Service Manger and sends the information entered by the administrator. A unique identifier (UUID) is generated by the Web Service Manager for each web service cluster started to distinguish it from the other web service clusters. The UUID is generated by using the current timestamp, the identifier of the server (machine) running the Web Service Manager and the encrypted string of the web service name. The web service cluster is identified by the UUID assigned to it.


2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
5. Delete a webservice cluster
6. Go back to the main menu
1
Enter the access point(s) of the genericSoapServer
Following genericServers are available
1. http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
2. http://btest.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
3. http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
Enter the numbers of the servers separated by commas. e.g 1,2
1,3
Enter the details for the web service
Enter the name to be entered in the Web service Registry:
A simple test web service
Enter the description
A web service to test the fault-tolerant system
Enter the keywords (separated by commas)
fault,tolerant,system
Enter the Document URL
http://btest.rnet.missouri.edu/webservices
Enter the access point(s) of the back-up genericSoapServer
Following genericServers are available
1. http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
2. http://btest.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
3. http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
Enter the numbers of the servers separated by commas. e.g 1,2
2
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

Web Service group created with the following responses for each generic path
NOTE: If response is a http path, it is the access point of the service started

  Generic Paths                                                                   Responses

1.http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl   http://darwin.rnet.missouri.edu:8009/soap/simpleServic
e.wsdl
2.http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl   http://beagle.rnet.missouri.edu:8009/soap/simpleServic
e.wsdl

Press ENTER to go back to the main menu
```

```
Connected to darwin.rnet.missouri.edu                SSH2 - aes128-cbc - hmac-md5 - none   142x42   NUM
```

Figure 4.4b: Information Entered to Start a Web Service Cluster

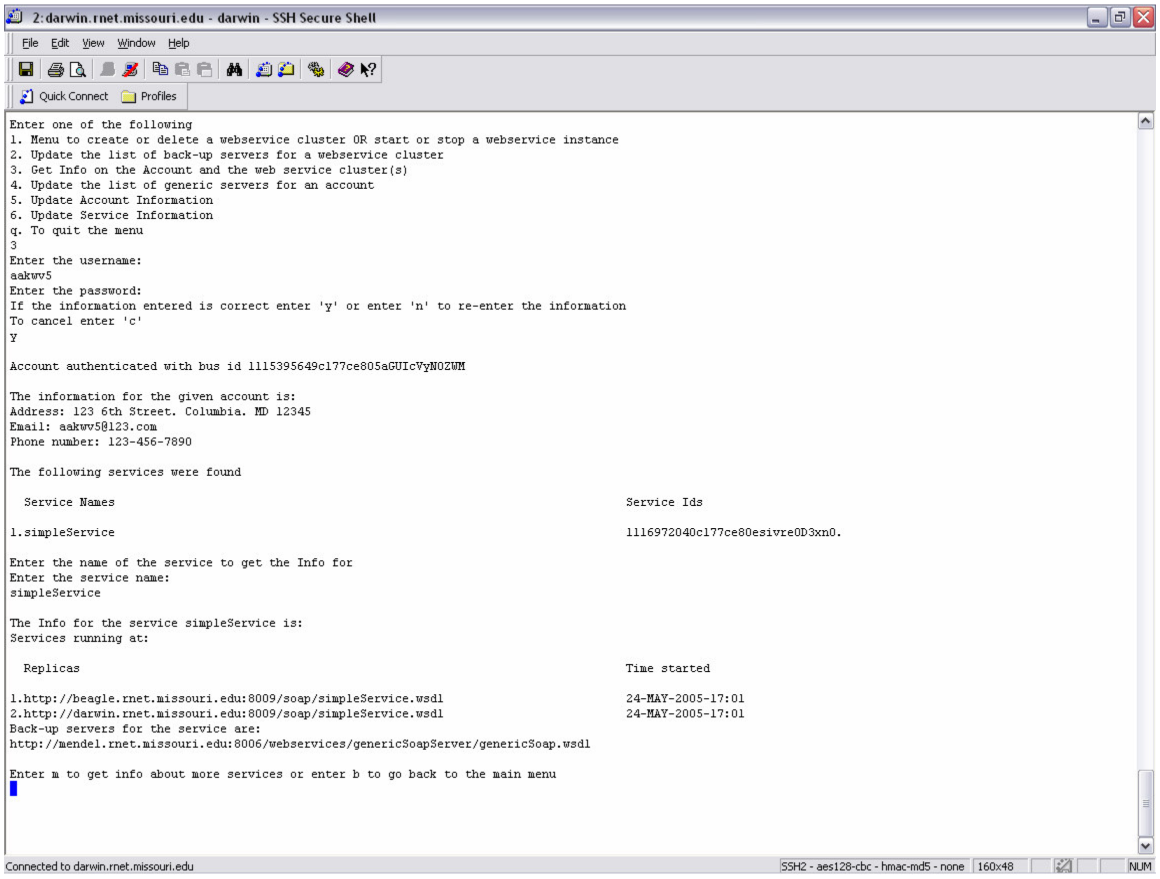
Since the request received from the Wsmclient is to start a web service cluster, the Web Service Manager calls the startService() function of the GenericSoapServers on the access point(s) provided by the administrator. The call to the GenericSoapServer is made using the GenericSoapClient. The parameters passed to the startService() function are the binary name of the web service (simpleService) and the UUID of the web service. As there might be more than one GenericSoapServer to be contacted, the calls to the GenericSoapServers are made simultaneously by forking a separate child process to call each GenericSoapServer. The GenericSoapServer on each server (machine) executes the binary (executable) of the ‘simpleService’ and enters the information in the local GDBM database. As mentioned in Section 3.5, the GenericSoapServer maintains the information about the web service instances started in a local GDBM database.

The GenericSoapServer(s) returns a message back to the Web Service Manager indicating whether or not the instance was successfully started. If the instance is successfully started, then the GenericSoapServer returns the access point of the web service instance that was started to the Web Service Manager. The Web Service Manager waits for a specified time interval to receive response back from all the GenericSoapServers that were contacted. Based on the response received from the GenericSoapServers, the Web Service Manager creates an update request to update the Oracle database with the web service cluster information. The web service cluster information includes the access point(s) of the web service instance(s) started by the GenericSoapServers, the access point(s) of the GenericSoapServers running on the back-up servers (machines), the UUID of the web service cluster, the UUID of the administrator and optional web service details. The Web Service Manager sends a multicast update message to the Listener(s) to update the database and receives a response back from the Listener(s). Based upon the response received by the Listener(s), the Web Service Manager notifies the administrator of the number of web service instances successfully started and their respective access points. Figure 4.4b shows that the response received for each GenericSoapServer called is the access point of the ‘simpleService’ instance started on that respective server (machine). Once a web service cluster is started, it is published in the Web Service Registry so that it is searchable by the clients.

After the web service cluster is started, the administrator might need to add a new instance to the web service cluster, stop an instance or just determine the status of one of the running instances. In order to execute these operations, the administrator should know the current information about the web service cluster as to on which servers (machines) the web service instances are running or the servers (machines) available to start the new instance. The following section describes how an administrator can retrieve information about a web service cluster from the Oracle database.

4.6 Retrieve Information about a Web Service Cluster

The Wsmclient can be used to retrieve information about a web service cluster. An administrator can administer more than one web service cluster and hence, the administrator might want to see the number of web service clusters he has started and retrieve information about each web service cluster. As shown in Figure 4.5, the third option provided by Wsmclient allows the administrator to see the list of the web service cluster(s) owned by the administrator. Before the list is shown to the administrator, the administrator is authenticated. The list shows the binary name of the web service cluster and the UUID of the web service cluster. The administrator's information is also shown along-with the list of the web service clusters owned by him. To retrieve details about a given web service cluster, the administrator is prompted to enter the binary name of that web service.


2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
160x48
NUM

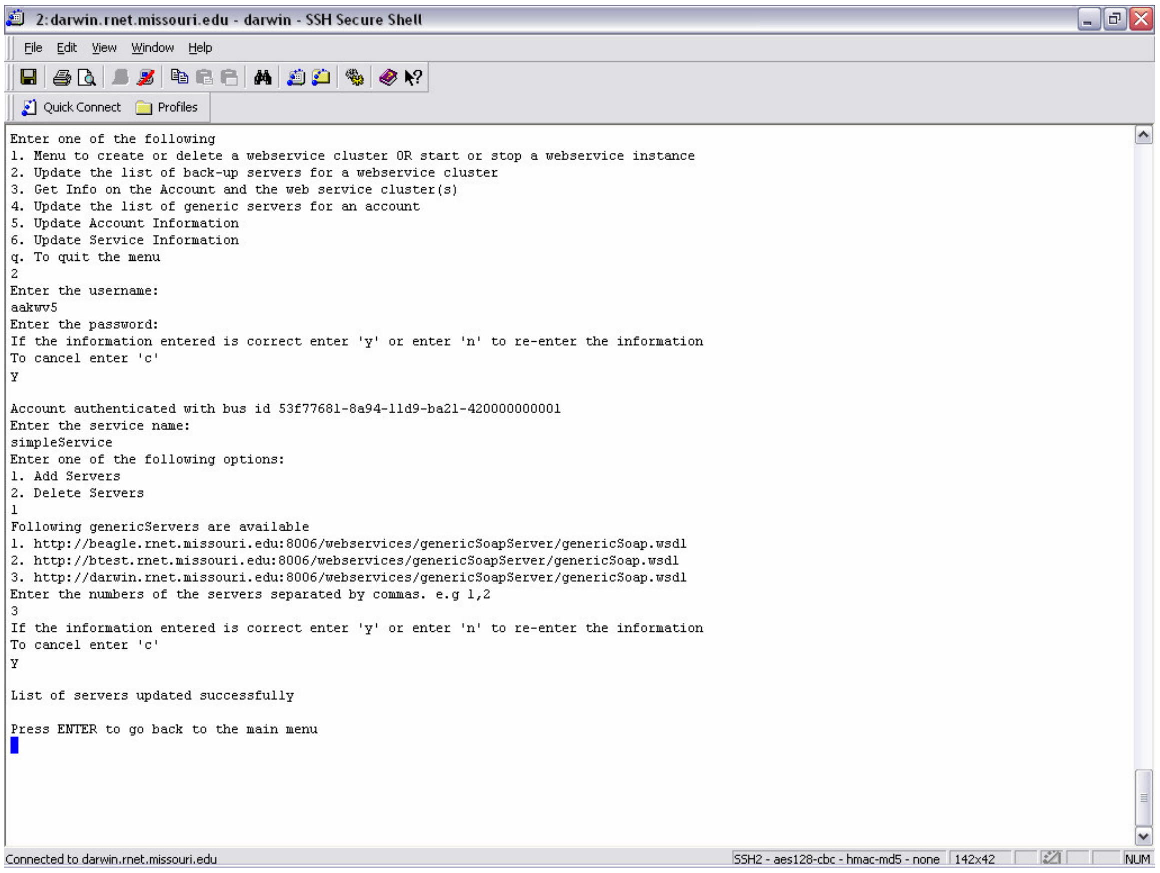
```
Enter one of the following
1. Menu to create or delete a webservice cluster OR start or stop a webservice instance
2. Update the list of back-up servers for a webservice cluster
3. Get Info on the Account and the web service cluster(s)
4. Update the list of generic servers for an account
5. Update Account Information
6. Update Service Information
q. To quit the menu
3
Enter the username:
aakwv5
Enter the password:
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

Account authenticated with bus id 1115395649c177ce805aGUIcVyNOZWM

The information for the given account is:
Address: 123 6th Street. Columbia. MD 12345
Email: aakwv5@123.com
Phone number: 123-456-7890

The following services were found

  Service Names                                                                         Service Ids

1.simpleService                                                                         1116972040c177ce80esivre0D3xn0.

Enter the name of the service to get the Info for
Enter the service name:
simpleService

The Info for the service simpleService is:
Services running at:

  Replicas                                                                              Time started

1.http://beagle.rnet.missouri.edu:8009/soap/simpleService.wsdl                          24-MAY-2005-17:01
2.http://darwin.rnet.missouri.edu:8009/soap/simpleService.wsdl                          24-MAY-2005-17:01
Back-up servers for the service are:
http://mendel.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl

Enter m to get info about more services or enter b to go back to the main menu
```

Figure 4.5: Information Displayed for a Web Service Cluster.

Figure 4.5 shows the information entered by the administrator to retrieve information for the ‘simpleService’ cluster. After the administrator has entered the web service name, the Wsmclient calls the Web Service Manager to retrieve information for that web service cluster. Before the Web Service Manager retrieves information for the web service cluster, it does an authorization check to see if the administrator requesting the information is the owner of the web service cluster. This is done to prevent an administrator from requesting information for a web service cluster not owned by him.

The Web Service Manager does the authorization check by remotely connecting to the database instance and checking if the UUID of the web service cluster is associated with the UUID of the administrator running Wsmclient. If the administrator owns the cluster, the Web Service Manger retrieves the information from the database for that given cluster and returns it to the Wsmclient to be displayed to the administrator. Figure 4.5 shows the information displayed for the web service cluster, ‘simpleService’, by the Wsmclient. It shows that there are two instances of the service running along-with the date they were started. It also shows the one access point of the GenericSoapServer running on the back-up server (machine) assigned to the web service cluster.

When a web service cluster is started, the cluster is assigned a set of back-up servers (machines), which are used by the Web Service Manager and the Monitor to start a new web service instance in the event of a failure of a running instance. Once a web service instance is started on a back-up server (machine), the back-up server’s (machine) name is removed from the back-up server’s list. Hence, it is necessary to periodically check and update the back-up server’s list for a web service cluster to ensure that there are back-up servers (machines) always available to the Web Service Manager and the Monitor to start new instances of a web service. The backup servers (machines) run the GenericSoapServers that are contacted by the Web Service Manager or the Monitor to start a web service instance. The access points of the GenericSoapServers are stored in the database and are designated as back-up servers (machines). The list of the access point(s) of the GenericSoapServers running on the back-up servers (machines) can be updated using the second option as shown in Figure 4.6. Updating the list of the access point(s) of the GenericSoapServers running on the back-up servers (machines) is similar to updating the list of the access point(s) of the GenericSoapServers as discussed in Section 4.4. Figure 4.6 shows the information entered to update the access point(s) list of the GenericSoapServers designated as back-up servers.


2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none 142x42 NUM

```
Enter one of the following
1. Menu to create or delete a webservice cluster OR start or stop a webservice instance
2. Update the list of back-up servers for a webservice cluster
3. Get Info on the Account and the web service cluster(s)
4. Update the list of generic servers for an account
5. Update Account Information
6. Update Service Information
q. To quit the menu
2
Enter the username:
aakwv5
Enter the password:
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

Account authenticated with bus id 53f77681-8a94-11d9-ba21-420000000001
Enter the service name:
simpleService
Enter one of the following options:
1. Add Servers
2. Delete Servers
1
Following genericServers are available
1. http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
2. http://btest.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
3. http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
Enter the numbers of the servers separated by commas. e.g 1,2
3
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

List of servers updated successfully

Press ENTER to go back to the main menu
```

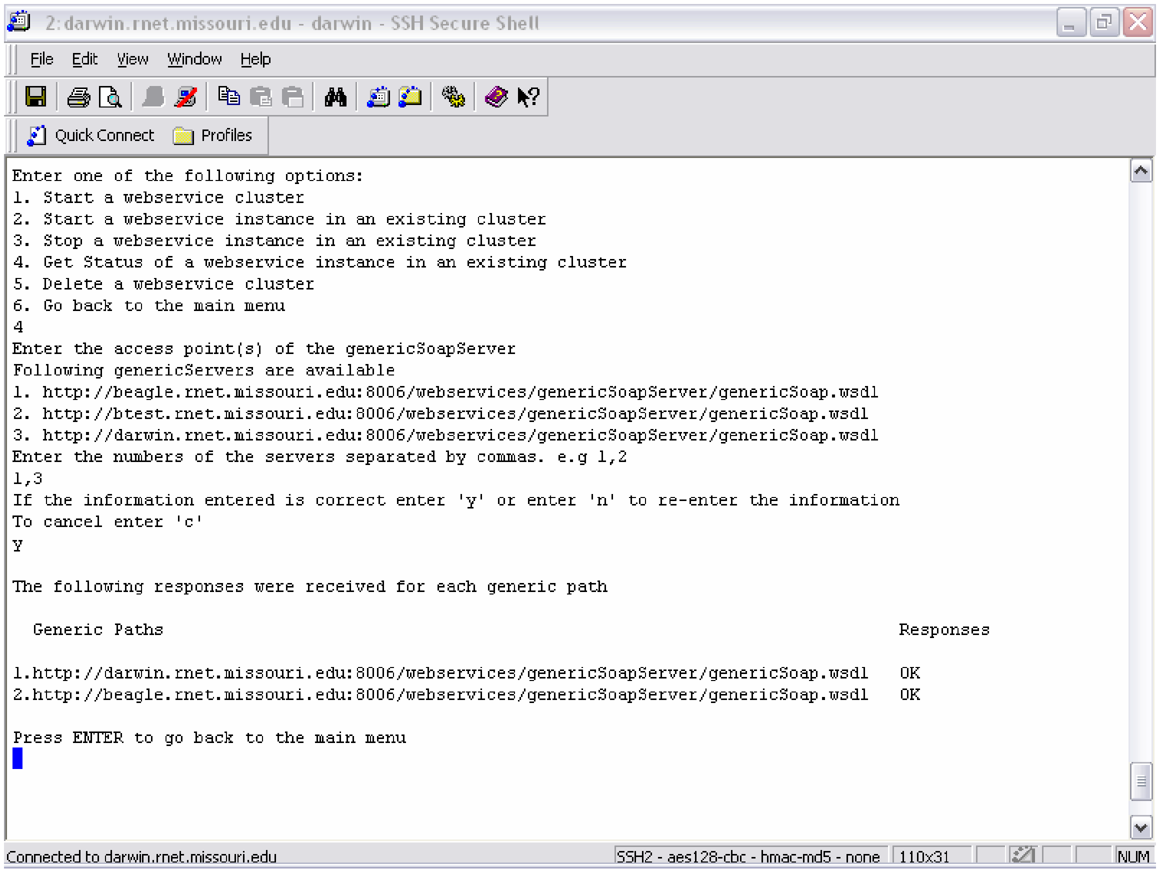
Figure 4.6: Information Entered to Update Back-up server List

After a web service cluster has been started, the administrator can start, stop or determine the status of a web service instance using Wsmclient. The next section discusses the procedures adopted to add (start) a new instance to an existing web service cluster, stop an instance and determine the status of a started instance.

4.7 Start, Stop and Determine Status of a Web Service Instance

In addition to starting and deleting a web service cluster, the Wsmclient also provides options to execute the three tasks of starting a web service instance, stopping and determining the status of a web service instance. This is shown in Figure 4.4a in Section 4.5. As shown in Figure 4.4a, before the second of options is shown, the administrator is asked if he wants to update the list of the access points and also prompted to enter the web service cluster name for which he needs to execute the second set of options. Identical information is needed to execute any of these three procedures. The information needed includes the name of the web service cluster (simpleService) and the access point(s) of the GenericSoapServers running on the servers (machines), where the web service instance is to be administered. The execution of these three procedures is similar and hence, only the procedure for determining the status of a running web service instance is discussed.

The information entered by the administrator in addition to the web service name is shown in Figure 4.7. The administrator is required to enter the access point(s) of the GenericSoapServers running on the servers (machines), where the web service instance is running. Instead of prompting the administrator to type in the access points, he is shown the GenericSoapServer's list to select the access point(s) from the list. The administrator can select more than one access point of the GenericSoapServer to determine the status of more than one web service instance within the same web service cluster. As shown in Figure 4.7, the administrator selects the access points of the GenericSoapServers running on the servers, *beagle.rnet.missouri.edu* and *darwin.rnet.missouri.edu*.


2: darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
110x31
NUM

```
Enter one of the following options:
1. Start a webservice cluster
2. Start a webservice instance in an existing cluster
3. Stop a webservice instance in an existing cluster
4. Get Status of a webservice instance in an existing cluster
5. Delete a webservice cluster
6. Go back to the main menu
4
Enter the access point(s) of the genericSoapServer
Following genericServers are available
1. http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
2. http://btest.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
3. http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
Enter the numbers of the servers separated by commas. e.g 1,2
1,3
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y

The following responses were received for each generic path

  Generic Paths                                                                       Responses

1.http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl   OK
2.http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl   OK

Press ENTER to go back to the main menu
```

Figure 4.7: Information Entered to Determine the Status of a Web Service Instance

After the information is gathered from the administrator, the Wsmclient makes a remote procedure call to the Web Service Manager and passes it the necessary information. The Web Service Manager does the authorization check to see if the web service cluster is owned by the administrator. The Web Service Manager retrieves the access point(s) of the web service instances to be checked from the Oracle database using the web service UUID and the access point(s) of the GenericSoapServers provided by the administrator. Since the administrator provided two access points running on two servers (machines), the Web Service Manager retrieves two access points of the web service instances running on those two servers (machines). The Web Service Manager tries to contact the two web service instances on the retrieved access points using GenericSoapClient. The Web Service Manager forks a child process to call each web service instance, so if more than one instance needs to be contacted, the calls are made simultaneously.

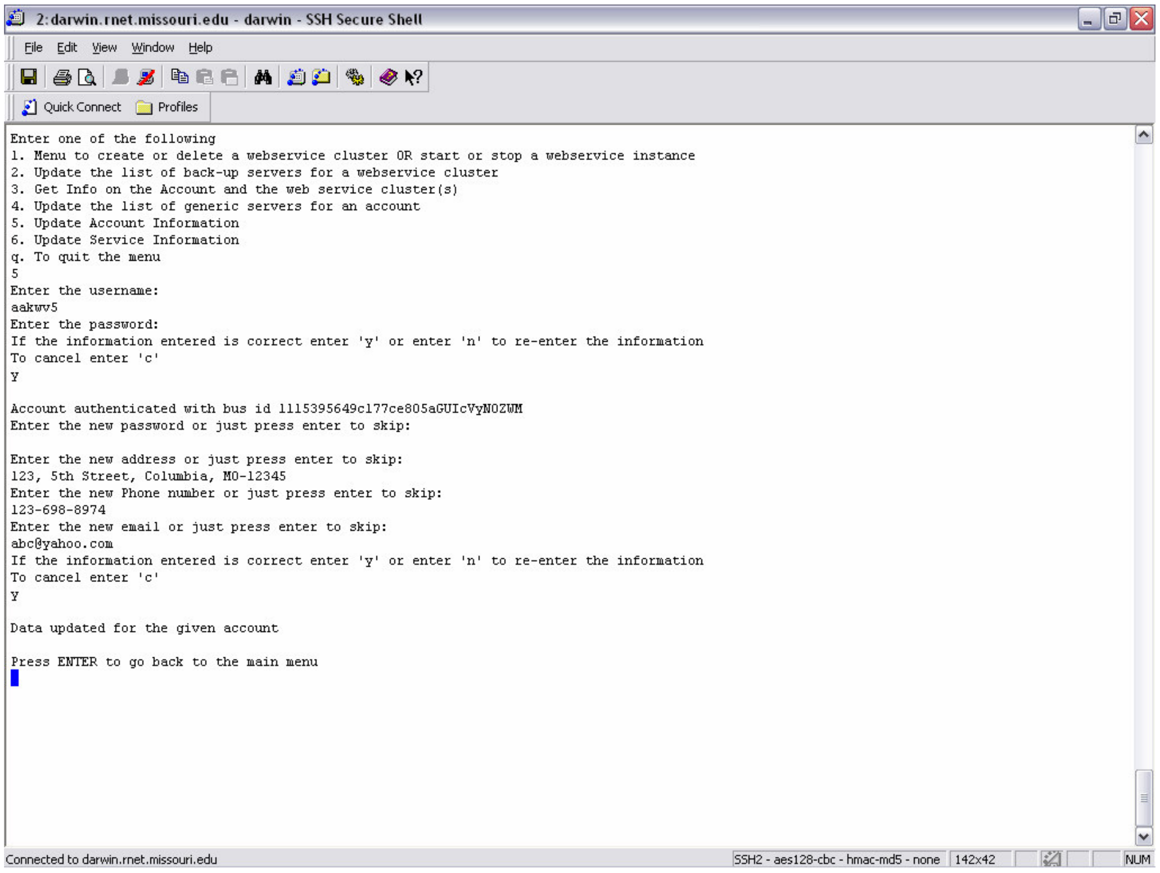
All the web service instances started as web service clusters implement a generic function called getStatus(). This function can be called to check the availability of a given web service instance. The Web Service Manager calls the getStatus() function of the web service instances to determine their availability. If the web service instance is available and functioning properly it returns an ‘OK’ response to the Web Service Manager. If the web service instance is not available, the Web Service Manager is not able to communicate with the web service, in which case the Web Service Manager times out after a specified time interval. If the web service instance is running properly, the Oracle database does not need to be updated. If the Web Service Manager times out contacting the web service instance, the Web Service Manger has to update the Oracle database to remove the access point of the web service instance from the database.

The Web Service Manager waits to receive responses from the two web service instances contacted. If it is not able to communicate with the web service instance(s), it forms an update request message to remove the web service instance(s) from the Oracle database and sends it on the multicast address to the Listener(s). The Web Service Manager sends an appropriate message back to the Wsmclient to be shown to the administrator indicating whether or not the web service instance(s) is available. As shown in Figure 4.7, an “OK” message is received from the two web service instances, indicating that both the instances are available on *beagle.rnet.missouri.edu* and *darwin.rnet.missouri.edu* respectively.

The account information provided when an administrator account is created by the super-administrator, or the web service information provided when a web service cluster is started by the administrator is used by the Web Service Registry to display information about the web service and the owner of the web service. There might be a need to change the account information or the service information to provide up-to-date information about the web service and the owner of the web service. The following section discusses the procedures to update the administrator's account information and the web service information.

4.8 Update Administrator Account and Web Service Information

The Wsmclient allows an administrator to change its account information such as the password, the email address, the mail address and the telephone number of the administrator by providing an option to update the account information as shown in Figure 4.8. Similarly, the web service information, such as the name of the web service to be displayed on the web registry, the service description, the service keywords and the service documentation page can be updated by using the option to update the service information as shown in Figure 4.8. The steps executed and the information entered to update both the account and the service information is similar and hence, only the procedure followed to update the account information is discussed.


2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
142x42
NUM

```
Enter one of the following
1. Menu to create or delete a webservice cluster OR start or stop a webservice instance
2. Update the list of back-up servers for a webservice cluster
3. Get Info on the Account and the web service cluster(s)
4. Update the list of generic servers for an account
5. Update Account Information
6. Update Service Information
q. To quit the menu
5
Enter the username:
aakwv5
Enter the password:
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
Y

Account authenticated with bus id 1115395649c177ce805aGUIcVyNOZWM
Enter the new password or just press enter to skip:

Enter the new address or just press enter to skip:
123, 5th Street, Columbia, MO-12345
Enter the new Phone number or just press enter to skip:
123-698-8974
Enter the new email or just press enter to skip:
abc@yahoo.com
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
Y

Data updated for the given account

Press ENTER to go back to the main menu
```

Figure 4.8: Information Entered to Update Administrator Account Information

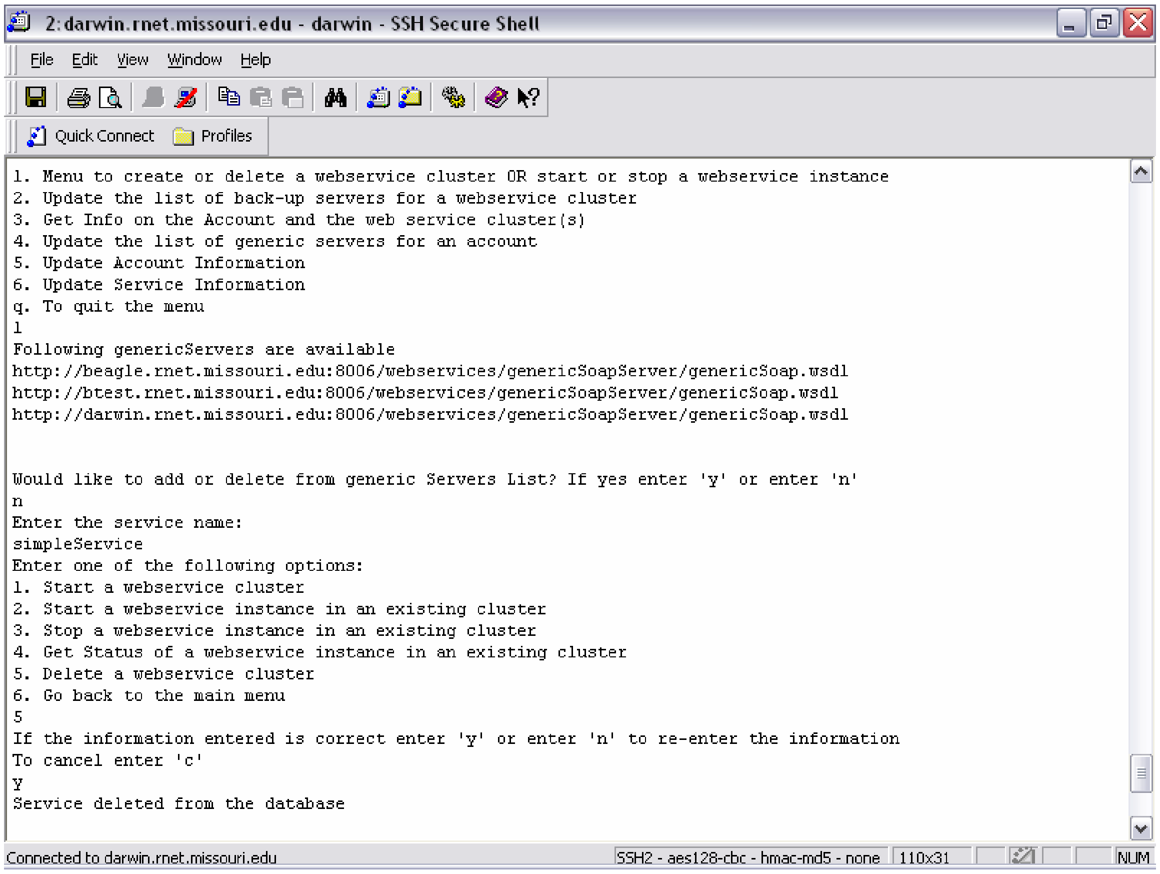
The administrator is authenticated before he is prompted to enter the account information needing to be updated in the Oracle database. As shown in Figure 4.8, the administrator is asked to enter the new value for each field he wants to update. The administrator can move forward to the next field by just pressing ENTER on the keyword if the administrator does not want to update that field. The Wsmclient calls the Web Service Manager passing the information to be updated. If the request is to update web service information, an authorization check is done to see if the administrator is the owner of the web service. The Web Service Manager updates the Oracle database by sending an update request message on the multicast address to the Listener(s). The Web Service Manager, upon receiving a response from the Listener(s), notifies the administrator with an appropriate message as shown in Figure 4.8.

The Oracle database maintains information for a web service cluster such as the name of the web service, the access point(s) of the instances of the service, the list of the back-up servers and other information. It is advisable to clean up the information from the database that is not meant to be used to prevent unnecessary resource consumption. Hence, if a web service cluster is not used or is not active, the data for the cluster should be removed from the Oracle database. The following section discusses the process to remove a web service cluster from the Oracle database.

4.9 Remove a Web Service Cluster

The Wsmclient can be used to remove information for a web service cluster from the database. The second set of options provided by the Wsmclient to administer a web service cluster, as shown in Figure 4.9, include an option to delete a web service cluster from the database. After the administrator is authenticated and before the second set of options is shown, the administrator is asked whether or not he wants to update the list of the access point(s) of the GenericSoapServers. In this case, since the procedure to be executed is to remove a web service cluster, the administrator does not update the GenericSoapServers list. The information needed by the

Wsmclient to remove a web service cluster from the Oracle database is the name of the web service.


2: darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
110x31
NUM

```
1. Menu to create or delete a webservice cluster OR start or stop a webservice instance
2. Update the list of back-up servers for a webservice cluster
3. Get Info on the Account and the web service cluster(s)
4. Update the list of generic servers for an account
5. Update Account Information
6. Update Service Information
q. To quit the menu
1
Following genericServers are available
http://beagle.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
http://btest.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl
http://darwin.rnet.missouri.edu:8006/webservices/genericSoapServer/genericSoap.wsdl


Would like to add or delete from generic Servers List? If yes enter 'y' or enter 'n'
n
Enter the service name:
simpleService
Enter one of the following options:
1. Start a webservice cluster
2. Start a webservice instance in an existing cluster
3. Stop a webservice instance in an existing cluster
4. Get Status of a webservice instance in an existing cluster
5. Delete a webservice cluster
6. Go back to the main menu
5
If the information entered is correct enter 'y' or enter 'n' to re-enter the information
To cancel enter 'c'
y
Service deleted from the database
```

Figure 4.9: Steps Followed to Remove a Web Service Cluster

As shown in Figure 4.9, the administrator enters the name of the web service (simpleService) which is to be removed from the database. After the user selects the option to delete a web service cluster, the Wsmclient calls the Web Service Manager and passes it the name of the web service cluster. The Web Service Manager checks to see if the web service is owned by the administrator. Before the Web Service Manager sends an update request message to the Listener(s) on the multicast address to remove the web service cluster information, it queries the database to find if there is any web service instance still running for that web service cluster. The information for a web service cluster cannot be removed from the database if there is a web service instance(s) running for that web service cluster. If the Web Service Manager finds an entry of an instance which is still running, a message is sent back to the administrator indicating that there is an instance running and the web service cluster cannot be removed from the database.

The administrator is required to stop the instance(s) using procedure described in Section 4.7, and then re-issue a request to remove the web service cluster. On the other hand, if there are no instances running for that web service cluster, the Web Service Manager removes the web service cluster information from the database by sending an update request message to the Listener(s). A message is shown to the administrator at the completion of the procedure as shown in Figure 4.9.

As explained in Section 3.6, a web service instance(s) tries to announce its failure to the Web Service Manager using WsmSoapServer. WsmSoapServer is a web service and it provides a web service interface to the Web Service Manager. The web service instances from other web service clusters can notify the Web Service Manager of their failure through WsmSoapServer. This prevents the servers (machines) hosting the web service instances from having to install DCE-RPC libraries to make RPC calls to the Web Service Manager. Since WsmSoapServer is a web service, it is started as a web service cluster just like any other web service. The cluster of the WsmSoapServer can be started following the procedure discussed in Section 4.2. Thus, starting the WsmSoapServer as a cluster is the last component to be deployed to provide a fault-tolerant system to start and administer fault-tolerant web services.

The presence of WsmSoapServer enables the web service instances from other clusters to notify the Web Service Manager of their failure. The presence of WsmSoapServer helps the discovery of a failed instance immediately. If the WsmSoapServer is not started, the web service instance(s) will not be able to notify the Web Service Manager of the failure, in which case, the Monitor will discover the failure sometime later. The Monitor is described in Section 3.4 and it runs periodically to check the availability of the web service instances. The following section discusses the interaction among the components when a web service instance notifies the Web

Service Manager of its failure through WsmSoapServer. The section uses the 'simpleService' web service instance.

4.10 Failure Communicated by a Web Service Instance

Figure 4.10 shows the steps and the interaction among the components of the fault-tolerant system to notify the Web Service Manager of a failure of a web service instance. It also shows the steps taken by the Web Service Manager upon receiving a notification of failure from a web service instance.

A web service instance notifies the Web Service Manager of its failure through the WsmSoapServer. Thus, the web service instance needs to know the access point of the WsmSoapServer to communicate with it. As mentioned earlier, the WsmSoapServer is started as a web service cluster and is also assigned a unique identifier (UUID), just like any other web service cluster. This UUID of the WsmSoapServer is used by the web service instance of any other cluster to get the access point of the WsmSoapServer from the Web Service Registry. The WsmSoapServer gets published with the Web Service Registry when WsmSoapServer is started as a web service cluster by the administrator.

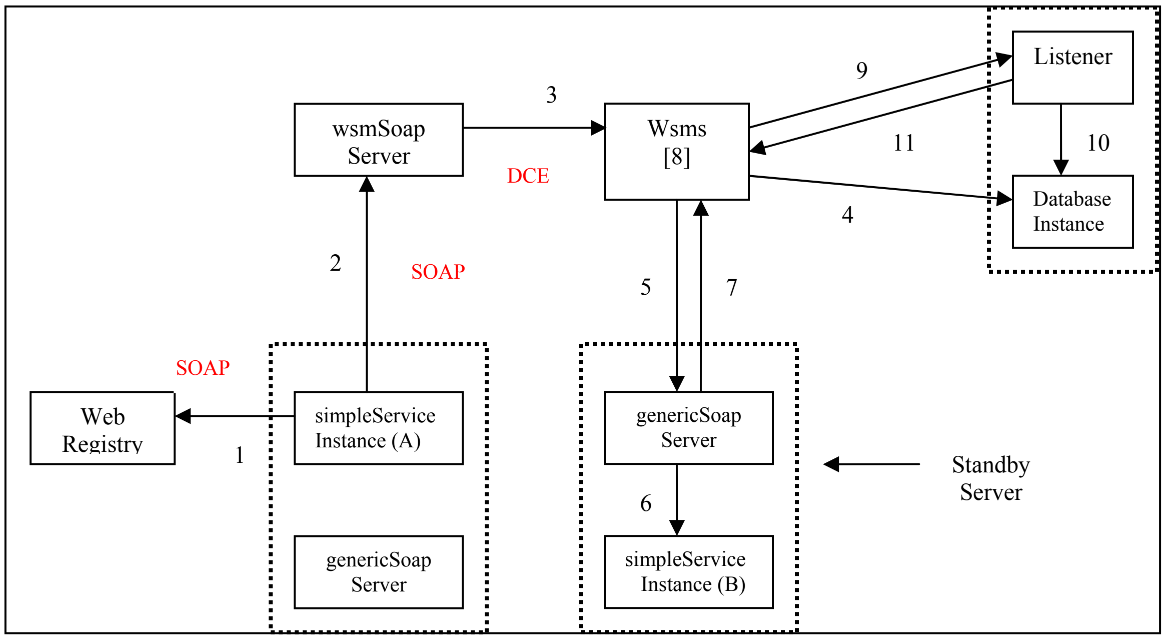


Figure 4.10: Notification of a Failure by a Web Service Instance to the Web Service Manager

After the web service instance of ‘simpleService’ is started, it contacts the Web Service Registry to get the access point (address) of the WsmSoapServer [step 1]. It contacts the Web Service Registry periodically to get the updated access points, in case the WsmSoapServer instance(s) has been moved to a different server(s) (machine). When the web service instance of ‘simpleService’ is about to fail, it calls a function called notify() of the WsmSoapServer running at the access point (address) fetched from the Web Service Registry [step 2]. This call to the WsmSoapServer may or may not be successful, based on the nature of the failure of the web service instance. It is possible that the web service instance might not have sufficient time to call the WsmSoapServer before the web service instance fails. If the web service instance is successful in calling the WsmSoapServer, the WsmSoapServer in turn calls the remote procedure of the Web Service Manager using DCE-RPC and passes it the access point of the web service instance that has failed and the name of the web service [step 3].

The Web Service Manager connects to the database instance and retrieves the access point of the GenericSoapServer running on a back-up server (machine), to start a new instance to replace the failed instance [step 4]. It then contacts the GenericSoapServer on the new server (machine) and instructs it to start a new instance of the web service ‘simpleService’ [step 5]. The GenericSoapServer starts a new instance of the web service ‘simpleService’ [step 6] and returns a message to the Web Service Manager, indicating whether or not the new instance was successfully started [step 7]. The Web Service Manager prepares an update request to update the information in the database such as removing the access point of the instance that failed and adding the access point of the new instance that was started [step 8]. The update request is sent on the multicast address to the Listener(s) [step 9], which updates the local database instance [step 10] and sends the response back to the Web Service Manager [step 11]. Thus, the database is updated so that the users get updated access points of the ‘simpleService’ from the Web Service Registry.

With the above description of the communication of failure of a web service instance to the Web Service Manager, we conclude the discussion of the usage of the fault-tolerant system to start and administer a web service cluster. The aforementioned procedures discussed in this chapter can be used to create any number of administrative accounts, and start any number of web service clusters. The following section gives a performance evaluation of the fault-tolerant system in terms of the execution time of the procedures to start and administer a web service cluster and, create and manage administrative accounts.

4.11 Performance Evaluation

The various procedures discussed earlier ranging from creating an administrator account to deleting a web service cluster were executed to evaluate the performance of the fault-tolerant system. The procedures were run a number of times daily, for a period of seven days and the execution time of each procedure along-with its CPU utilization was noted. The execution time is defined as the time from start to stop of a program. The CPU utilization of a program is the CPU time devoted to the program by a server (machine) for the program’s execution. The two servers (machines) where the fault-tolerant system was deployed are Compaq Tru64 UNIX V5.1B, 4 processor systems.

The various procedures discussed earlier are categorized into two categories based on the number of the components of the fault-tolerant system that participate in the execution of the procedures. The discussion begins with the category that includes procedures to create and manage administrative accounts. The components that are involved in the execution of these procedures are Ws_admin, Web Service Manager and Listener (database instance).

4.11.1 Evaluation of Procedures to Create and Manage Administrator Account

As discussed earlier in Section 4.3, the program used to create and manage an administrator's account is Ws_admin. The Ws_admin accepts input from the super-administrator and contacts the Web Service Manager to create a new administrator account or retrieve information about the administrator accounts. The Web Service Manager, in turn, updates or retrieves information about the accounts from the Oracle database using the Listener(s) running on the servers (machines) where the Oracle database instance(s) are running. The updates to the Oracle database are performed by the Web Service Manager by sending an update request message on the multicast address to the Listener(s). The Web Service Manager performs a read operation by remotely connecting to one of the available Oracle database instances.

The procedures executed by Ws_admin to create and manage administrator accounts are further divided into the ones that initiate a read operation and the ones that initiate a write (update) operation on the Oracle database through the Web Service Manager. The procedures such as creating an account, deleting an account and resetting an account require the database to be modified and hence, initiate a write (update) operation on the database. The procedure just to list the administrative accounts initiates a read operation on the Oracle database.

The write (update) operations require the Web Service Manager to send a multicast message to the Listener(s) and receive a response back from the Listener(s) about the update operation. The read operation requires the Web Service Manager to establish a remote connection to the database instance and retrieve the data. The procedures that initiate a write operation take a longer time to execute than the procedures that initiate a read operation because of the additional multicast message sent on the network to update the database instance(s).

Table 4.1 shows the statistics of the execution times along-with the CPU utilization of the procedures to create and administer administrator accounts. Table 4.1 shows the minimum time taken, the maximum time taken and the average time taken to execute the procedures. As mentioned earlier, the procedures were executed daily for a span of seven days. The procedures were run three times daily at different times during the day. The average is computed over 21 runs for each of the procedures. All the tables discussed further in this section follow the same format.

Procedures	Operation	Average (CPU : Execution)	Minimum (CPU : Execution)	Maximum (CPU : Execution)
Creating account	Write	0.022 sec : 2.3 sec	0.016 sec : 1 sec	0.033 sec : 3 sec
Deleting account	Write	0.020 sec : 2.2 sec	0.016 sec : 1 sec	0.033 sec : 4 sec
Resetting account	Write	0.025 sec : 2.3 sec	0.016 sec : 1 sec	0.033 sec : 3 sec
List accounts	Read	0.020 sec : 1.04 sec	0.016 sec : 1 sec	0.033 sec : 2 sec

Table 4.1: Statistics of the Procedures to Manage an Administrator Account

Table 4.1 shows the CPU utilization of Ws_admin on its local server (machine) and the total execution time on the fault-tolerant system to execute the respective procedure. If the procedure initiates a write operation, the total execution time of the procedure on the fault-tolerant system includes the time spent in updating a database instance(s) through multicast. The time spent in updating a database instance(s) depends upon the time needed for a multicast message to reach the Listener from the Web Service Manager, the time taken to execute the update request on the database instance by the Listener and the time to receive a response back from the Listener by the Web Service Manager. Table 4.2 shows the statistics of the execution times needed in updating the database instance by the Web Service Manager through multicast. From the Table 4.1 and Table 4.2, we can infer that the execution time taken for procedures that initiate a write operation approximately exceeds the procedures that initiate a read operation by the additional time needed to do a multicast update.

	Average (Execution)	Minimum (Execution)	Maximum (Execution)
Multicast Update	2.15 sec	1 sec	3 sec

Table 4.2: Statistics of the Multicast Update to Manage an Administrator Account

The next category discusses procedures needed to start and administer a web service cluster besides doing just a database read or a write operation. It involves the interaction of one more component of the fault-tolerant system, the GenericSoapServer. Thus, there are four components that participate in the execution of these procedures; the Wsmclient, the Web Service Manager, the GenericSoapServer and the Listener (database instance).

4.11.2 Evaluation of Procedures to Administer a Web Service Cluster

The execution times of the procedures to administer a web service cluster differ from the ones discussed in the previous section, primarily because they involve one more component of the fault-tolerant system (GenericSoapServer) and secondarily, the multicast update for these operations takes a longer time to execute. As mentioned earlier, the time taken for a multicast update depends upon the time taken by the Listener to execute the update request on the database instance and the round trip communication between the Web Service Manager and the Listener.

The time taken to execute an update request on the database instance by the Listener varies based upon the number of tables needing to be updated in the database and whether or not indexes on some of the tables in the database need to be rebuilt. Indexes are built on some of the tables in the database that hold information about a web service such as its name, description and other information. This information is used by the Web Service Registry while searching for a web service based on a search criteria provided by the user. The indexes need to be rebuilt if any information in those tables is updated. Thus, if the update request modifies information in the tables that have indexes built on them, the indexes need to be rebuilt to accommodate the change.

When a web service cluster is started, the Web Service Manager can be instructed to start more than one instance of the web service. The Web Service Manager forks a child process to contact a GenericSoapServer to administer a web service instance. Thus, the Web Service Manager forks as many child processes as there are GenericSoapServers needing to be contacted.

The CPU utilization and the execution times of the procedures have been noted for starting a web service cluster with a different number of web service instances ranging from one to three. The statistics were also noted for simultaneously stopping web service instances ranging from one to three through the Web Service Manger. Table 4.3 shows the statistics for starting a web service cluster with web service instances ranging from one to three. The table shows the CPU utilization and the total execution time of the Wsmclient and, the CPU utilization and the execution time of the Web Service Manager to communicate with the GenericSoapServer. Table 4.4 shows the statistics for stopping web service instances through the Web Service Manager.

	Average (CPU : Execution)	Minimum (CPU : Execution)	Maximum (CPU : Execution)
With one instance			
Wsmclient	0.020 sec : 41 sec	0.016 sec : 35 sec	0.033 sec : 55 sec
Web Service Manager to communicate with GenericSoapServer	4.6 sec : 29 sec	4.4 sec : 25 sec	5 sec : 33 sec
With two instances			
Wsmclient	0.020 sec : 42.3 sec	0.016 sec : 38 sec	0.033 sec : 56 sec
Web Service Manager to communicate with GenericSoapServer	9.37 sec : 28.4 sec	9.4 sec : 25 sec	10.5 sec : 33 sec
With three instances			
Wsmclient	0.020 sec : 48.2 sec	0.016 sec : 43 sec	0.033 sec : 57 sec
Web Service Manager to communicate with GenericSoapServer	14.9 sec : 32.6 sec	14.5 sec : 31 sec	15.7 sec : 35 sec

Table 4.3: Statistics of the Procedure to Start a Web Service Cluster

	Average (CPU : Execution)	Minimum (CPU : Execution)	Maximum (CPU : Execution)
One instance			
Wsmclient	0.020 sec : 10 sec	0.016 sec : 9 sec	0.033 sec : 16 sec
Web Service Manager to communicate with GenericSoapServer	4.6 sec : 7.6 sec	4.4 sec : 7 sec	5 sec : 10 sec
Two instances			
Wsmclient	0.020 sec : 10.8 sec	0.016 sec : 10 sec	0.033 sec : 14 sec
Web Service Manager to communicate with GenericSoapServer	9.9 sec : 7.9 sec	9.6 sec : 7 sec	10.5 sec : 9 sec
Three instances			
Wsmclient	0.020 sec : 12 sec	0.016 sec : 10 sec	0.033 sec : 15 sec
Web Service Manager to communicate with GenericSoapServer	14.7 sec : 9.5 sec	14.4 sec : 8 sec	15.5 sec : 13 sec

Table 4.4: Statistics of the Procedure to Stop a Web Service Instance(s)

From Tables 4.3 and 4.4, we observe that the total execution time of Wsmclient is greater to start a web service cluster with a given number of instances compared to stopping the same number of web service instances, whereas the CPU utilization of Wsmclient is approximately the same. The CPU utilization is the same because in both the cases, the work done by the CPU on the server (machine) where Wsmclient is running is to call Web Service Manager and wait for a response back from the Web Service Manager. The total execution time of Wsmclient depends upon the time taken by the Web Service Manger to communicate with the GenericSoapServers, to update the database through multicast and return a message back to Wsmclient. As evident from Table 4.3 and Table 4.4, the execution time for a round trip communication of the Web Service Manager with the GenericSoapServer for starting a cluster with a given number of web service instances is greater compared to stopping the same number of web service instances. This is because it takes more time for the GenericSoapServer to start a web service instance on the local server (machine) than to stop a web service instance.

As discussed in Section 3.5, the GenericSoapServer has to locate the binary of the web service instance and then fork a process to start the web service instance. The GenericSoapServer waits until the web service instance has started and upon the successful execution of the web service instance, the GenericSoapServer returns the access point of the web service instance to the Web Service Manager. On the other hand, the GenericSoapServer does not have to do much work to stop a web service instance. To stop a web service instance, it retrieves the process identifier of the web service instance and stops the process. The time taken by the GenericSoapServer to stop a web service instance is less than to start a web service instance. Thus, it takes more time for the Web Service Manager to receive a response back from the GenericSoapServer.

On the other hand, the CPU utilization time of the Web Service Manager for starting and stopping a given number of web service instances is almost the same. This is because the Web Service Manager forks one child process per each GenericSoapServer to be contacted. Hence, the number of child processes forked to start a cluster with a given number of web service instances is same to stopping the same number of web service instances.

The other factor that contributes to the difference in execution times of the Wsmclient to start and stop a given number of web service instances is the time taken by the Web Service Manger to update the database through multicast. Table 4.5 shows the statistics of the execution time of a database update by the Web Service Manager through multicast for the procedures to administer a web service cluster for a given number of web service instances. The execution time of a multicast update varies based upon the data to be updated and whether or not the indexes on some of the tables in the database need to be rebuilt. The execution time is less when no index is needed to be rebuilt whereas the execution time is greater when the indexes are required to be rebuilt.

Multicast Update	Average (Execution)	Minimum (Execution)	Maximum (Execution)
One instance			
Without index(s) rebuilt	2.2 sec	1 sec	6 sec
With index(s) rebuilt	12 sec	8 sec	16 sec
Two instances			
Without index(s) rebuilt	2.5 sec	2 sec	5 sec
With index(s) rebuilt	9.8 sec	8 sec	13 sec
Three instances			
Without index(s) rebuilt	2.5 sec	2 sec	5 sec
With index(s) rebuilt	11 sec	8 sec	15 sec

Table 4.5: Statistics of the Multicast Update to Administer a Web Service Cluster

Starting a web service cluster entails updating the Oracle database with information about the web service such as its name, description and keywords associated with the web service. This information is used in searching for a web service in the web service registry. Thus, starting a web service cluster involves rebuilding of the indexes on the tables that store the service information. On the contrary, stopping a web service instance does not require the aforementioned service information to be removed from the database. This is because only the web service instance(s) is stopped and not the web service cluster removed from the database. Hence, the indexes on the tables that store the relevant service information are not needed to be rebuilt when a web service instance is stopped. Since starting a web service cluster requires rebuilding the indexes, the execution time of a multicast update is greater than it is when a web service instance is stopped.

The execution time of a multicast update with indexes rebuilt from Table 4.5 for a given number of instances when added to the execution time of the Web Service Manager to communicate with the GenericSoapServer from Table 4.3 for the same number of instances, it gives approximately the total execution time of Wsmclient to start a web service cluster with that many number of instances. The same observation is true for stopping a web service instance when the execution time of a multicast update without indexes rebuilt is used from Table 4.5.

If you compare the statistics of Table 4.3 internally for different number of instances, we observe that the statistics are approximately the same except for the CPU utilization of the Web Service Manager. The CPU utilization of the Web Service Manager for starting a web service cluster with two instances is approximately twice the CPU utilization for stating a web service cluster with one instance. This is because the Web Service Manager forks two child processes to start two web service instances whereas the Web Service Manager forks only one child process to start one web service instance. For starting a web service cluster with three instances the CPU utilization is thrice of what it is for one instance. Thus, the greater is the number of child processes forked, the greater is the CPU utilization. The statistics of the CPU utilization in Table 4.7 differ internally for different number of web service instances for the same reason.

The statistics of Table 4.5 are approximately the same for different number of instances. This shows that the multicast update for a respective procedure, whether to start or stop a web service instance(s) is same regardless of the number of the web service instances involved.

The Wsmclient also provides an option to delete a web service cluster from the database. It must be executed only after all the web service instances in that cluster have been stopped. Table 4.6 shows the statistics for deleting a web service cluster from the database. Deleting web service cluster information involves rebuilding the indexes in the database for the required tables.

	Average (CPU : Execution)	Minimum (CPU : Execution)	Maximum (CPU : Execution)
Delete a web service cluster	0.020 sec : 10.6 sec	0.016 sec : 8 sec	0.033 sec : 15 sec

Table 4.6: Statistics of the Procedure to Delete a Web Service Cluster

Comparing the statistics from all the tables above, we reach the following conclusions.

1) The total execution time of the Wsmclient and the time taken by the Web Service Manager to communicate with the GenericSoapServers to start a web service cluster is approximately the same regardless of the number of web service instances. This is because the GenericSoapServers are contacted simultaneously to start the web service instances.
2) The total execution time of the Wsmclient to stop a given number of web service instances simultaneously is approximately the same.
3) The CPU utilization of the Web Service Manager depends upon the number of GenericSoapServers it has to communicate with. It takes approximately 4.5 – 5 sec of CPU utilization to communicate with one GenericSoapServer. Thus, the CPU utilization of the Web Service Manager is proportional to the number of GenericSoapServers needing to be contacted.

Starting a web service cluster is not the end of the story. It has to be advertised so that users on the World Wide Web (WWW) can find and use the web service. The next chapter (Chapter 5) discusses the Web Service Registry, a repository of information about the web services that provides a means to publish and search a web service.

5. Web Service Registry

The discovery of a web service is among the important concerns in the field of web services. Making a web service known to the world is crucial for the usage of the web service. A web service built and deployed, but not advertised is equivalent to the web service not existing. If the web service is built to be used by a large clientele, it needs to be appropriately advertised so that the clientele can easily find and utilize the web service. This thesis implements a Web Service Registry (WSR), which is a tool used by the developers and the clientele of the web services. Using the WSR the developers can publish their web services to advertise them to the world and the clients can search for those web services, based on some search criteria.

The WSR stores information about a web service, such as the location of the WSDL file that describes the web service, the documentation URL, and also information about the owner of the web service. This way, companies do not need to know anything about each other before they discover the services offered by other companies. The WSR provides an interface for the users to publish and retrieve information about web services. All the operations related to publishing a web service with the WSR are categorized as Admin operations, and the operations performed for searching or retrieving information about a web service are categorized as Inquire operations.

The WSR can be made available in two interfaces. A CGI interface that can be accessed using a web browser and a web service interface that can be accessed using a web service client program. The web service interface of the WSR consists of two web services. Thus, the WSR can be accessed with the help of web pages via a CGI interface using a web browser, or programmatically via a web service interface using a web service client. This is shown in Figure 5.1.

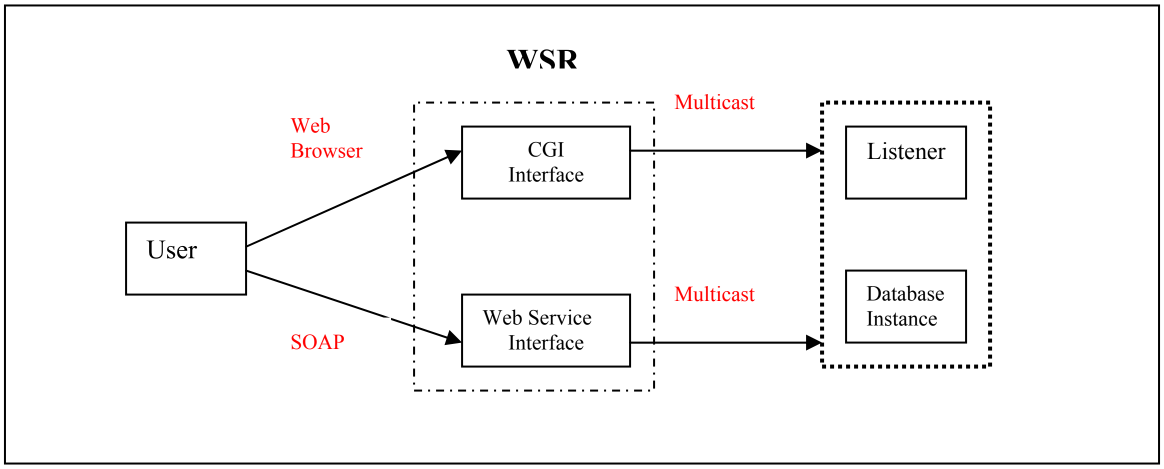


Figure 5.1: The Two Interfaces of the Web Service Registry

This thesis implements both types of interfaces. The WSR also uses the same Oracle database instance(s) used by the Web Service Manager and the Monitor. Thus, the programs that form the WSR interface use multicast to update the database instance(s) as shown in Figure 5.1. The programs contact the Listener(s) to get the address of the database instance to do a read operation. The servers (machines) running the WSR interface must have a Oracle client installed, for the programs of the WSR interface to remotely connect to the Oracle database instance(s) on other servers (machines). The database tables (schema) used by the WSR to store information in the database is available in Appendix A.

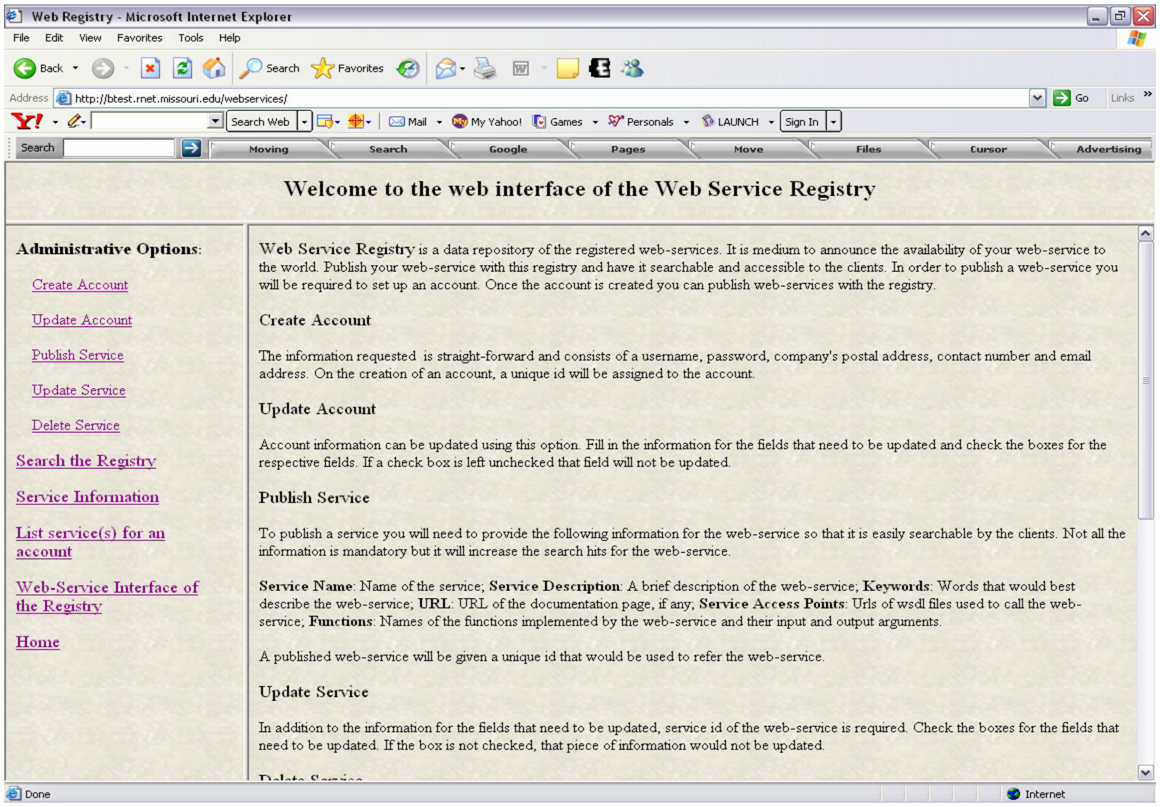
Any web service started as a web service cluster using the fault-tolerant system and the procedures discussed in Chapter 3 and Chapter 4 respectively, automatically gets registered with the WSR. Chapter 3 discusses the components of the fault-tolerant system and Chapter 4 discusses the procedures to use the fault-tolerant system to start and administer a web service cluster. The two web services of the web service interface of the WSR are also started as web service clusters using the fault-tolerant system.

The first two sections of this chapter discuss the implementation and the usage of the interfaces, CGI interface and web service interface. The last section demonstrates the usage of the

WSR by citing an example of an application web service published with the WSR and a client application utilizing the web service. The discussion begins with the CGI interface of the WSR.

5.1 Common Gateway Interface (CGI) Interface of the Web Service Registry

The CGI interface of the Web Service Registry gives a user the ability to interact with the WSR using a web browser. The main page of the CGI interface is shown in Figure 5.2 and can be accessed at http://btest.rnet.missouri.edu/webservices. The CGI interface is deployed on more than one server (machine) to achieve fault tolerance. The CGI interface is run on servers (machines) at a set of default URLs and can be started on new servers (machines) at new URLs. A Universal Resource Locator (URL) is the address of an entity on the World Wide Web (WWW). At-least one URL among the set of default URLs of the WSR must be available all the time to achieve fault-tolerance. The URL(s) of the CGI interface of the WSR is stored in the Oracle database instance(s). If the CGI interface is started at a new URL, it is entered in the Oracle database so that the users can get the new URL(s) of the interface besides the default URL(s) from the database as explained shortly.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Games Personals LAUNCH Sign In
Search Moving Search Google Pages Move Files Cursor Advertising
Done Internet

Welcome to the web interface of the Web Service Registry

Administrative Options:

- Create Account
- Update Account
- Publish Service
- Update Service
- Delete Service

Search the Registry

Service Information

List service(s) for an account

Web-Service Interface of the Registry

Home

Web Service Registry is a data repository of the registered web-services. It is medium to announce the availability of your web-service to the world. Publish your web-service with this registry and have it searchable and accessible to the clients. In order to publish a web-service you will be required to set up an account. Once the account is created you can publish web-services with the registry.

Create Account

The information requested is straight-forward and consists of a username, password, company's postal address, contact number and email address. On the creation of an account, a unique id will be assigned to the account.

Update Account

Account information can be updated using this option. Fill in the information for the fields that need to be updated and check the boxes for the respective fields. If a check box is left unchecked that field will not be updated.

Publish Service

To publish a service you will need to provide the following information for the web-service so that it is easily searchable by the clients. Not all the information is mandatory but it will increase the search hits for the web-service.

Service Name: Name of the service; **Service Description**: A brief description of the web-service; **Keywords**: Words that would best describe the web-service; **URL**: URL of the documentation page, if any; **Service Access Points**: Urls of wsdl files used to call the web-service; **Functions**: Names of the functions implemented by the web-service and their input and output arguments.

A published web-service will be given a unique id that would be used to refer the web-service.

Update Service

In addition to the information for the fields that need to be updated, service id of the web-service is required. Check the boxes for the fields that need to be updated. If the box is not checked, that piece of information would not be updated.

Figure 5.2 Main Page of the CGI Interface of the Web Service Registry

The clients need to know the URL(s) of the CGI interface to access the CGI interface. One way to make the URL(s) of the CGI interface available to the clients is to advertise the URL(s) of the CGI interface on a popular website, which is very frequently accessed, such as the company's or the institution's website. A disadvantage of this approach is that the popular website needs to be updated each time when a URL is added or deleted from the list of available URLs of the CGI interface of the WSR. Another way is to have the client retrieve the URL(s) of the CGI interface from the Oracle database. Retrieving the URL(s) from the Oracle database is a comparatively better approach as the clients get updated URL(s) of the interface from the Oracle database. To achieve this, a multicast client program is required, which contacts the Listener(s) on the multicast address and obtains the updated URL(s) of the interface from the database. A program called GetInterfaceUrl, which is written in JAVA, is built for this purpose. The source code is available in Appendix C.

The program GetInterfaceUrl tries to contact a Listener(s) on the multicast address and requests the updated address(s) of the CGI interface of the WSR. If the program receives a response back from the Listener, it displays the URL(s) (address(s)) of the CGI interface(s) available to the user. The program may not be able to communicate with the Listener due to a number of reasons, such as the network not supporting multicast or the routers in the path are not multicast enabled. When the program is not able to communicate with the Listener, the program displays a set of default URL(s) (address(s)) available to the user that are hard-coded in the GetInterfaceUrl program. As mentioned earlier, at-least one of the URL among the set of default URLs must be available all the time. This ensures that any client using the multicast program to retrieve URL(s) of the CGI interface is displayed at-least one available URL when the multicast program fails to communicate with the Listener(s).

The GetInterfaceUrl program is made available to the clients on a popular website, such as the company's or the institution's website. Once the program is downloaded, the user can use the program to fetch the updated URL(s) of the interface from the database. The user does not need to visit the popular website (company's or institution's website) again after downloading the program. Thus, even if the popular website is temporarily not available the user can get the URL(s) of the CGI interface by running the multicast program.

The CGI interface provides a means to publish a web service and search the WSR to find a web service. Publishing a web service on the WSR requires data to be updated in the Oracle database and hence, has to be executed in a secure environment with a way to authenticate the user. This is done by creating accounts for the user accessing the WSR and authenticating them based on a username and password. Searching the WSR is performed using read operations on the Oracle database and does not require the user to be authenticated. The operations needed to publish a web service are termed as Admin operations, and the operations to search the WSR as Inquire operations. Thus, the CGI interface of the WSR is further divided into an Admin interface and an Inquire interface. The Admin and the Inquire interfaces are discussed below.

5.1.1 Admin Interface

The options provided by the Admin Interface of the WSR are to create an account, to update an account, to publish a web service, to update a web service and to delete a web service. Each of the options is available in the form of links on the main page of the WSR as shown previously in Figure 5.2.

The generalized execution of each Admin operation on the WSR is illustrated in Figure 5.3. The links on the main page of the WSR direct the user to a HTML [2] form which is displayed in the web browser. A HTML form is a form which gathers information from the user and sends it to a remote program on the WWW. The user enters the information in the HTML form displayed in the web browser [step 1]. When the user submits the form, a specific program is called on a remote server (machine) using the CGI mechanism to execute the necessary operation [step 2]. The remote programs that execute the Admin operations read from and write to the Oracle database instance(s). The programs do a read operation by remotely connecting to the database instance [step 3]. A write operation is performed by sending an update request on the multicast message to the Listener(s) [step 4]. After the operation has been executed, the remote program returns the results back to the web browser to be displayed to the user [step 5]. The source code of the remote programs that execute the Admin operations is available in Appendix C.

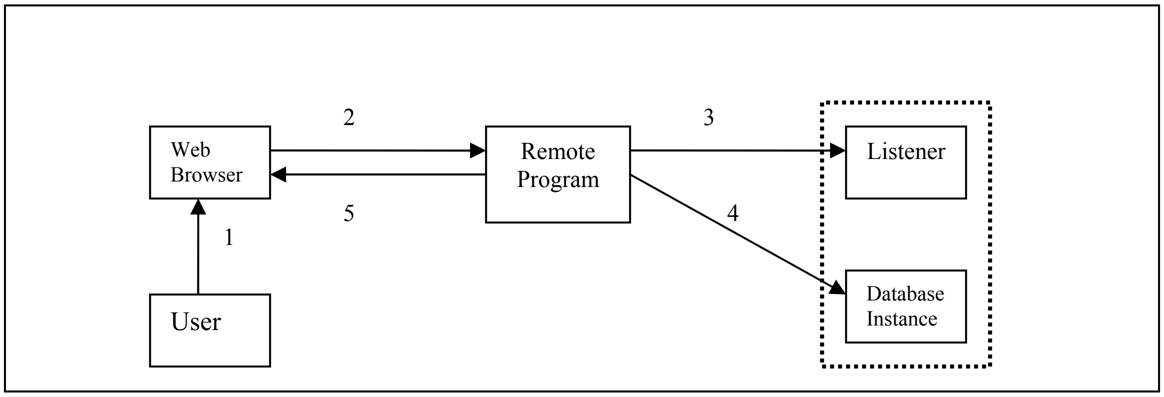


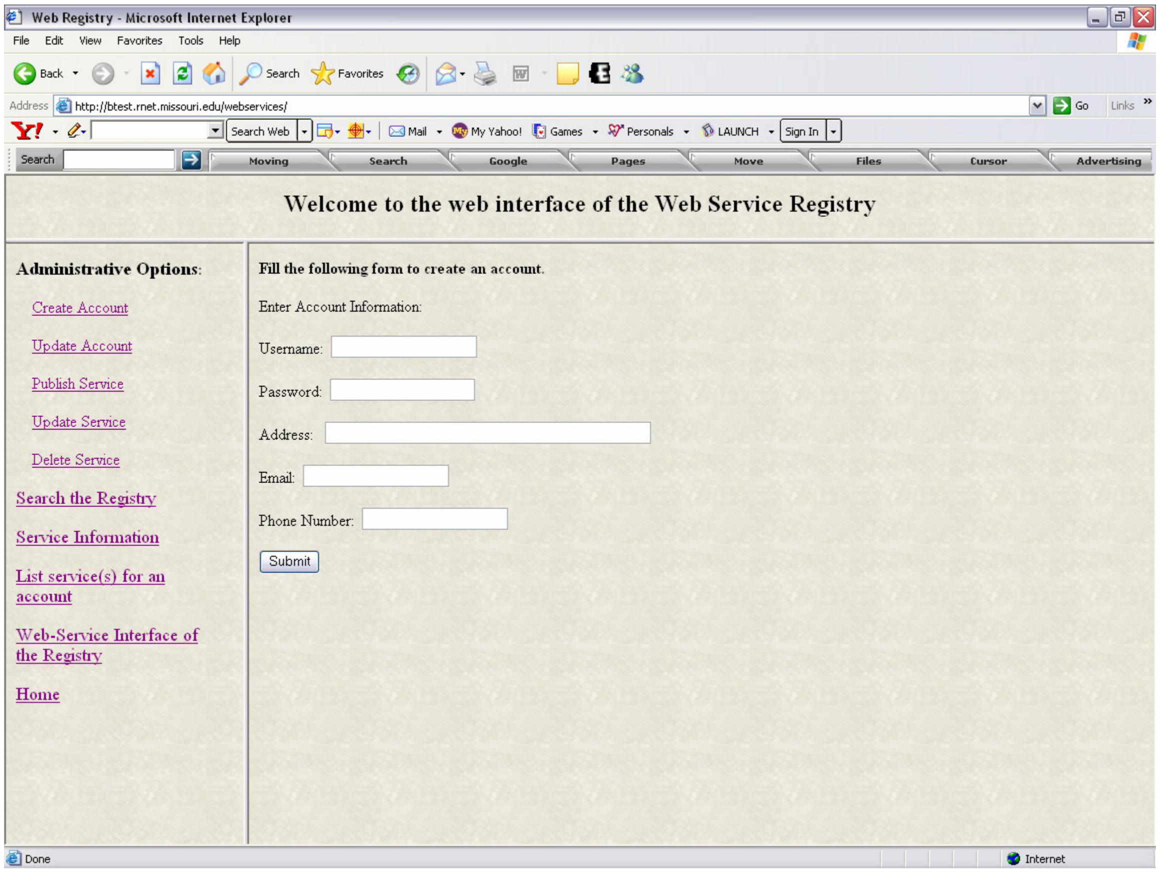
Figure 5.3: Execution of an Admin Operation on the CGI Interface of the Web Service Registry

In order to execute Admin operations on the WSR, the user is required to be authenticated. This requires the user to create an account with the WSR before any Admin operations are performed by the user. The accounts created with the WSR are called WSR accounts. The process adopted to create and update a WSR account is discussed below.

5.1.1.1 Create and Update a Web Service Registry Account

Any client who wants to publish a web service with the WSR needs to create a user account to access the Admin services of the WSR. The information needed to create an account includes the username, the password, the company address, the email address and the phone number. The company address, the email address and the phone number are displayed by the WSR along with the web service information whenever web service information is displayed on the WSR. This information is useful for the clients of the web service to contact the publisher of the web service for technical guidance and other help.

The 'Create Account' link under the administrative options on the main page of the WSR takes the user to a HTML form as shown in Figure 5.4. When the user has finished entering the information and submits the HTML form, the HTML form calls the CreateAccount program on the remote server, passing the required information. The program parses the information sent, encrypts the password and writes the information in a text file. The program does not create an account instantly. The CreateAccount program writes the information in a text file instead of sending a multicast update message to the Listener(s) to update the Oracle database with the new account information. This is done to prevent abuse of the create account capability on the web registry interface. If the accounts are created instantly without any supervision then anyone can misuse the capability to create multiple unwanted accounts and thus, consume the resources such as network bandwidth and the disk space. The requests to create WSR accounts by the users are written in a text file called requestAccount. The user is shown a message saying that he will receive a notification via an email when the WSR account is created.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://btest.rnet.missouri.edu/webservices/

Welcome to the web interface of the Web Service Registry

Administrative Options:

- Create Account
- Update Account
- Publish Service
- Update Service
- Delete Service
- Search the Registry
- Service Information
- List service(s) for an account
- Web-Service Interface of the Registry
- Home

Fill the following form to create an account.

Enter Account Information:

Username:

Password:

Address:

Email:

Phone Number:

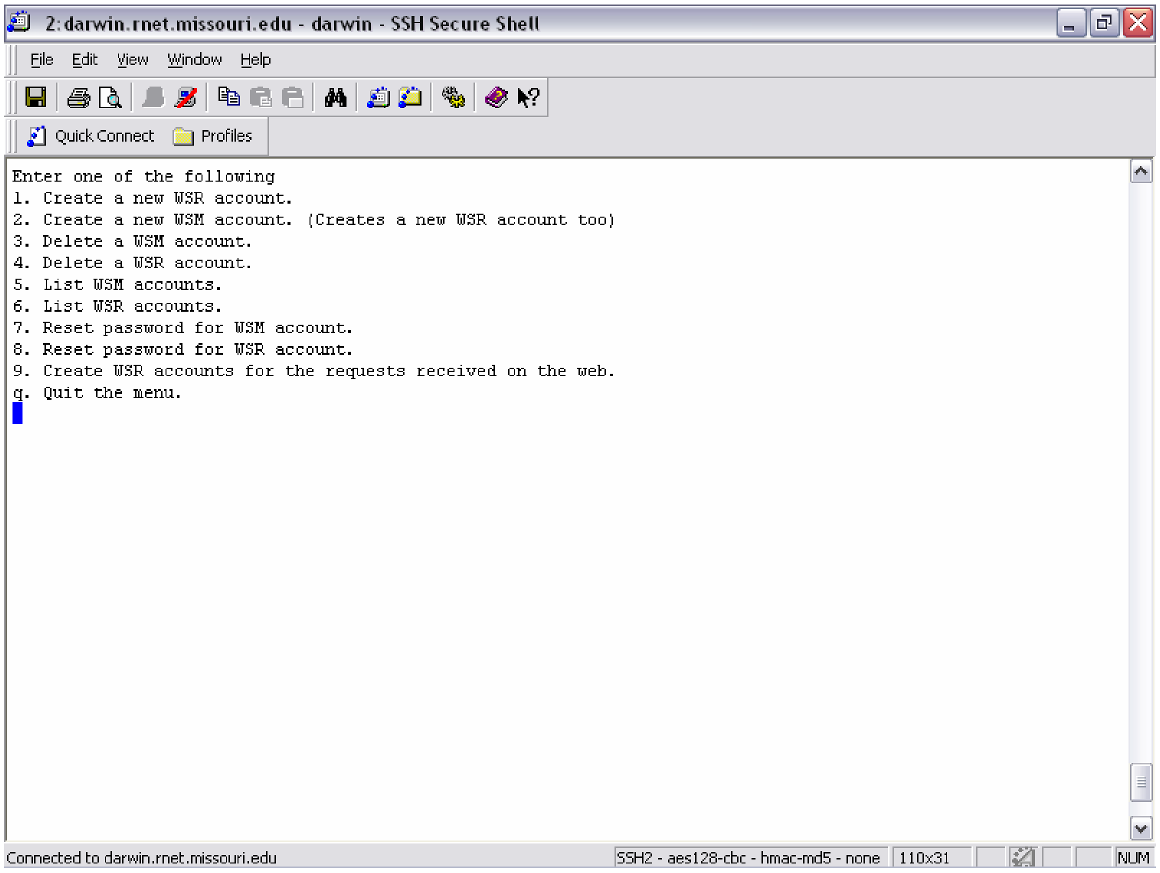
Submit

Figure 5.4: HTML Form to Create a Web Service Registry Account

The super-administrator who is responsible for creating the WSR accounts verifies the identity of the users requesting accounts. The identity of a user can be verified using the email address or the company address or the phone number of the user. Once the user information is verified, the super-administrator creates the WSR accounts by using Ws_admin program. The super-administrator is responsible to create WSR accounts to provide access to the Admin operations of the WSR. The usage of Ws_admin to create and manage WSR accounts is discussed below. The Ws_admin is also used to create administrator's account to access the fault-tolerant system to start and administer a web service cluster as discussed in Chapter 4 (Section 4.2).

Figure 5.5 shows the options provided by the Ws_admin to create WSR accounts. The first, fourth, sixth, eighth and ninth options allow the super-administrator to create, delete, reset and list the WSR accounts respectively.

The execution of the first, fourth, sixth and eighth option to create and manage WSR accounts is similar to those of administrator accounts discussed in Chapter 4 (Section 4.2). Using the first option, the super-administrator can create one WSR account at a time and he is required to enter the information for each account. This is not practical when there are a large number of requests to create WSR accounts. As mentioned earlier, all the requests for WSR accounts are written to a text file called requestAccount. The ninth option of Ws_admin allows the super-administrator to create WSR accounts for the requests submitted to the text file requestAccount.

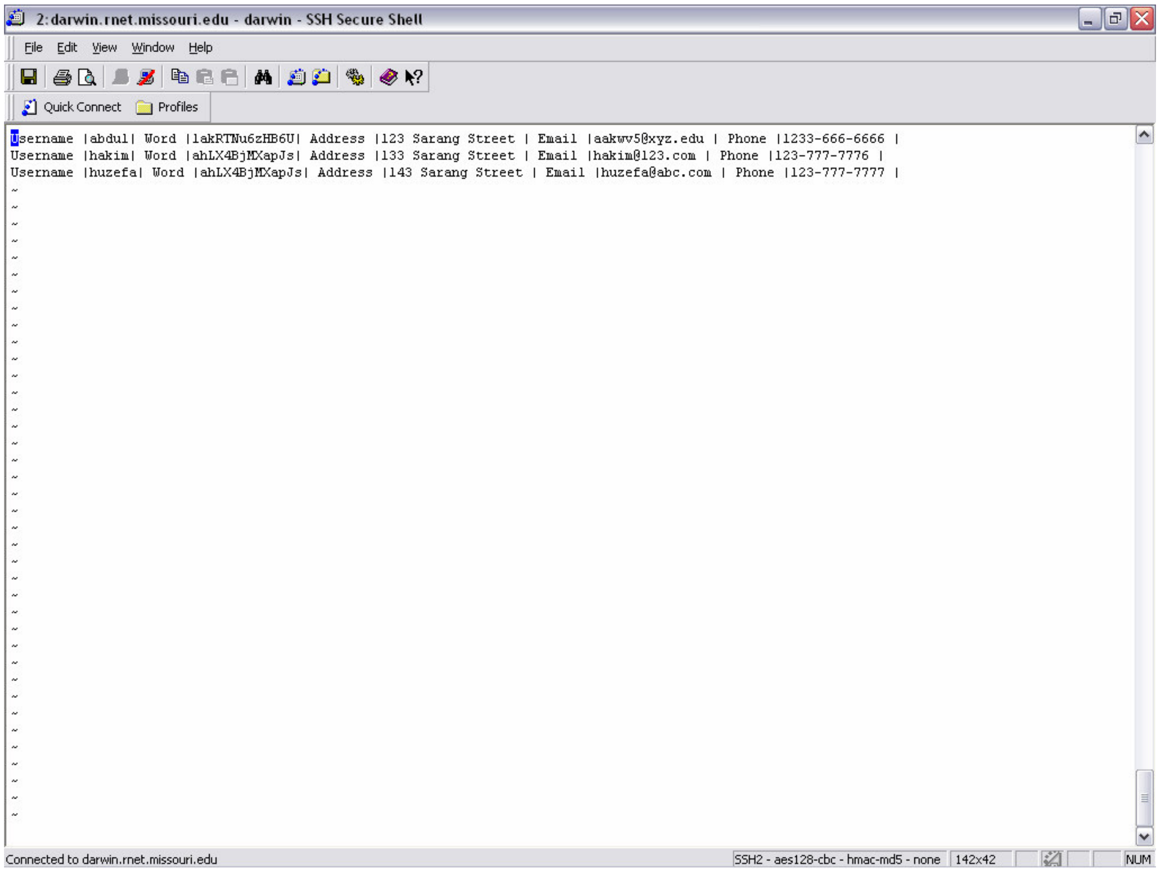

2: darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
Enter one of the following
1. Create a new WSR account.
2. Create a new WSM account. (Creates a new WSR account too)
3. Delete a WSM account.
4. Delete a WSR account.
5. List WSM accounts.
6. List WSR accounts.
7. Reset password for WSM account.
8. Reset password for WSR account.
9. Create WSR accounts for the requests received on the web.
q. Quit the menu.
```

Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
110x31
NUM

Figure 5.5: Options Offered by Ws_admin to Manage Web Service Registry Accounts

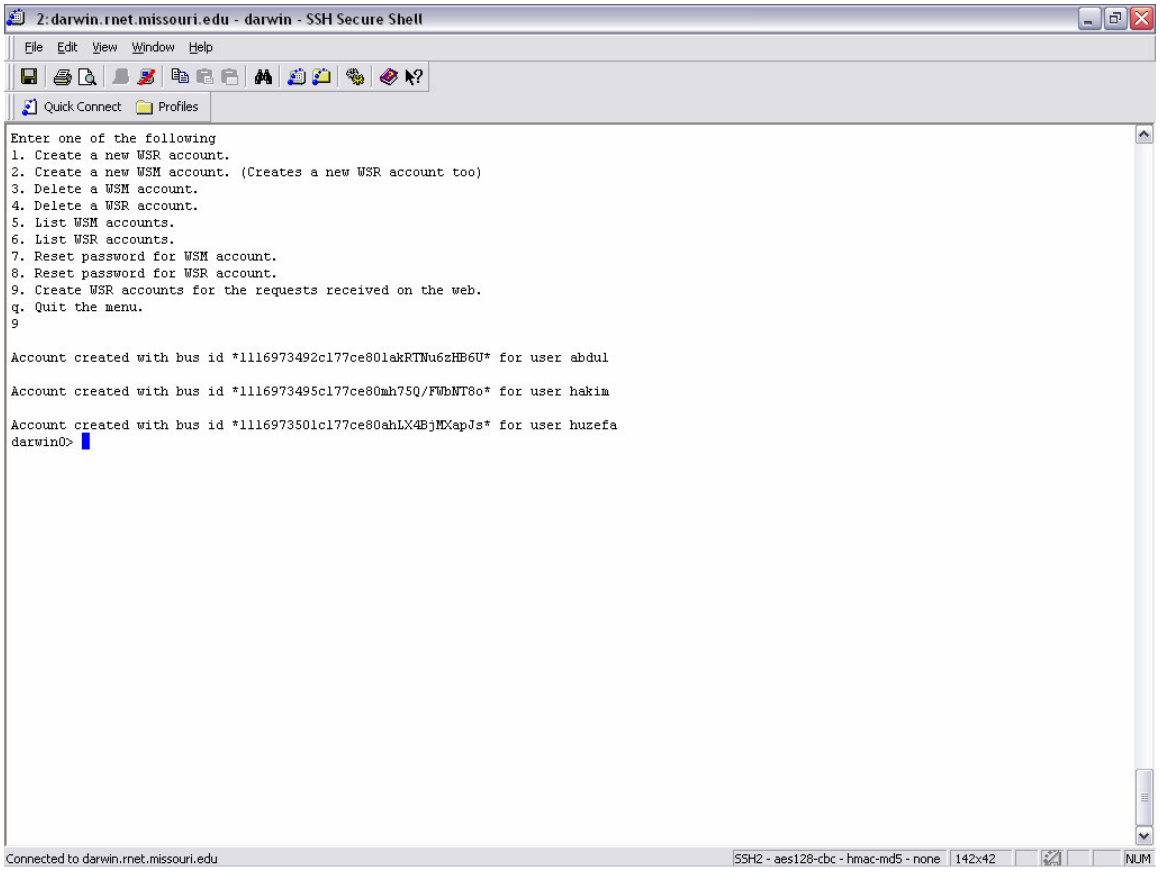
Let us suppose there have been three requests submitted to the WSR to create WSR accounts. The requestAccount file with the three requests is shown in Figure 5.6a. Each line in the file has information about one request received to create a WSR account. It has the username, password, address, email, and phone number of the user requesting a WSR account. The super-administrator verifies the identity of each user by using the information of each user in the text file. The identity can be verified based on the company address, the email address and the phone number. The super-administrator edits the text file to retain the lines of the users whose identities have been verified. The super-administrator deletes the lines of the users whose identities cannot be verified.


2: darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
142x42
NUM

```
Username |abdul| Word |lakRTNu6zHB6U| Address |123 Sarang Street | Email |aakwv5@xyz.edu | Phone |1233-666-6666 |
Username |hakim| Word |ahLX4BjMXapJs| Address |133 Sarang Street | Email |hakim@123.com | Phone |123-777-7776 |
Username |huzefa| Word |ahLX4BjMXapJs| Address |143 Sarang Street | Email |huzefa@abc.com | Phone |123-777-7777 |
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Figure 5.6a: Requests Submitted to Create Web Service Registry Accounts

Upon selecting the ninth option displayed by Ws_admin, as shown in Figure 5.6b, the information is read from the file requestAccount and accounts are created for each request in the file. The Ws_admin knows the location of the file requestAccount. The Ws_admin calls the Web Service Manager and passes the information collected from the file. The Web Service Manager generates a universally unique identifier (UUID) for each new WSR account created and inserts the information in the Oracle database using multicast. A UUID is created by concatenating the timestamp, the server (machine) identifier on which the Web Service Manager is running and the encoded string of the user's username. These three values ensure that the identifier generated for each WSR account is unique. Upon successful creation of an account, the super-administrator is displayed the UUID of the accounts created. Figure 5.6b shows the messages displayed after the accounts have been successfully created. Once the WSR accounts are created, the Ws_admin automatically sends an email to the users whose accounts have been created confirming the creation of a WSR account to access the Admin operations of the WSR.

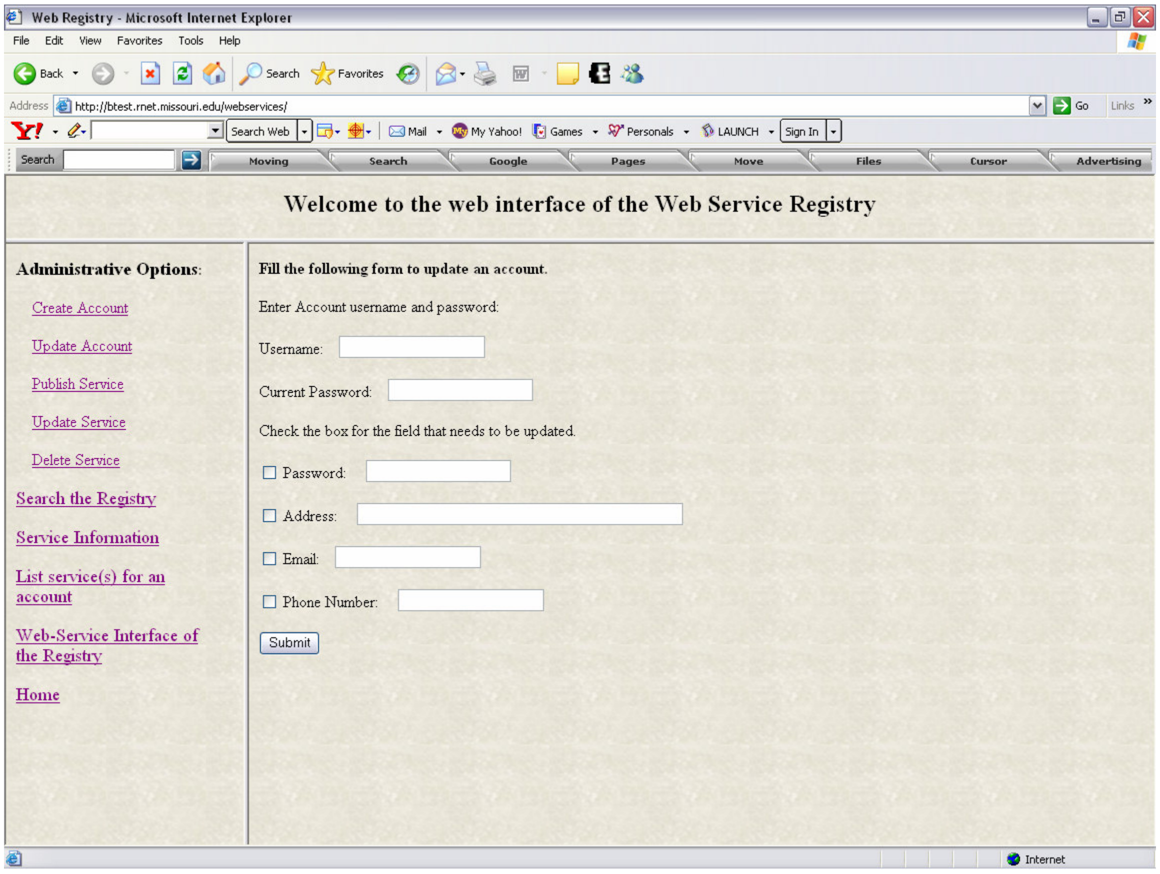

2:darwin.rnet.missouri.edu - darwin - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

```
Enter one of the following
1. Create a new WSR account.
2. Create a new WSM account. (Creates a new WSR account too)
3. Delete a WSM account.
4. Delete a WSR account.
5. List WSM accounts.
6. List WSR accounts.
7. Reset password for WSM account.
8. Reset password for WSR account.
9. Create WSR accounts for the requests received on the web.
q. Quit the menu.
9

Account created with bus id *1116973492c177ce801akRTNu6zHB6U* for user abdul

Account created with bus id *1116973495c177ce80mh75Q/FWbNT8o* for user hakim

Account created with bus id *1116973501c177ce80ahLX4BjMXapJs* for user huzefa
darwin0> 
```

Connected to darwin.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
142x42
NUM

Figure 5.6b: Execution of Ws_admin to Create Web Service Registry Accounts

The user might want to change his account information after he has been created an account on the WSR. The CGI interface also provides a means to update the WSR account information via the HTML form shown in Figure 5.7. It is displayed in the web browser by following the ‘Update Account’ link under the administrative options from the main page of the WSR. The user can select the fields he wants to update by checking the box next to those fields. Only the checked fields are updated. The user cannot change the username assigned to him. After the form is submitted, a call is made to the program updateAccount on the remote server. The program authenticates the user based on the username and the current password by doing a read operation from the Oracle database. If the user is authenticated, the account information is updated by sending an update request message on the multicast address to the Listener(s). The user gets an appropriate response back from the program, which is displayed in the web browser based upon whether or not the update was successful.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links

Welcome to the web interface of the Web Service Registry

Administrative Options:

Create Account

Update Account

Publish Service

Update Service

Delete Service

Search the Registry

Service Information

List service(s) for an account

Web-Service Interface of the Registry

Home

Fill the following form to update an account.

Enter Account username and password:

Username:

Current Password:

Check the box for the field that needs to be updated.

☐ Password:

☐ Address:

☐ Email:

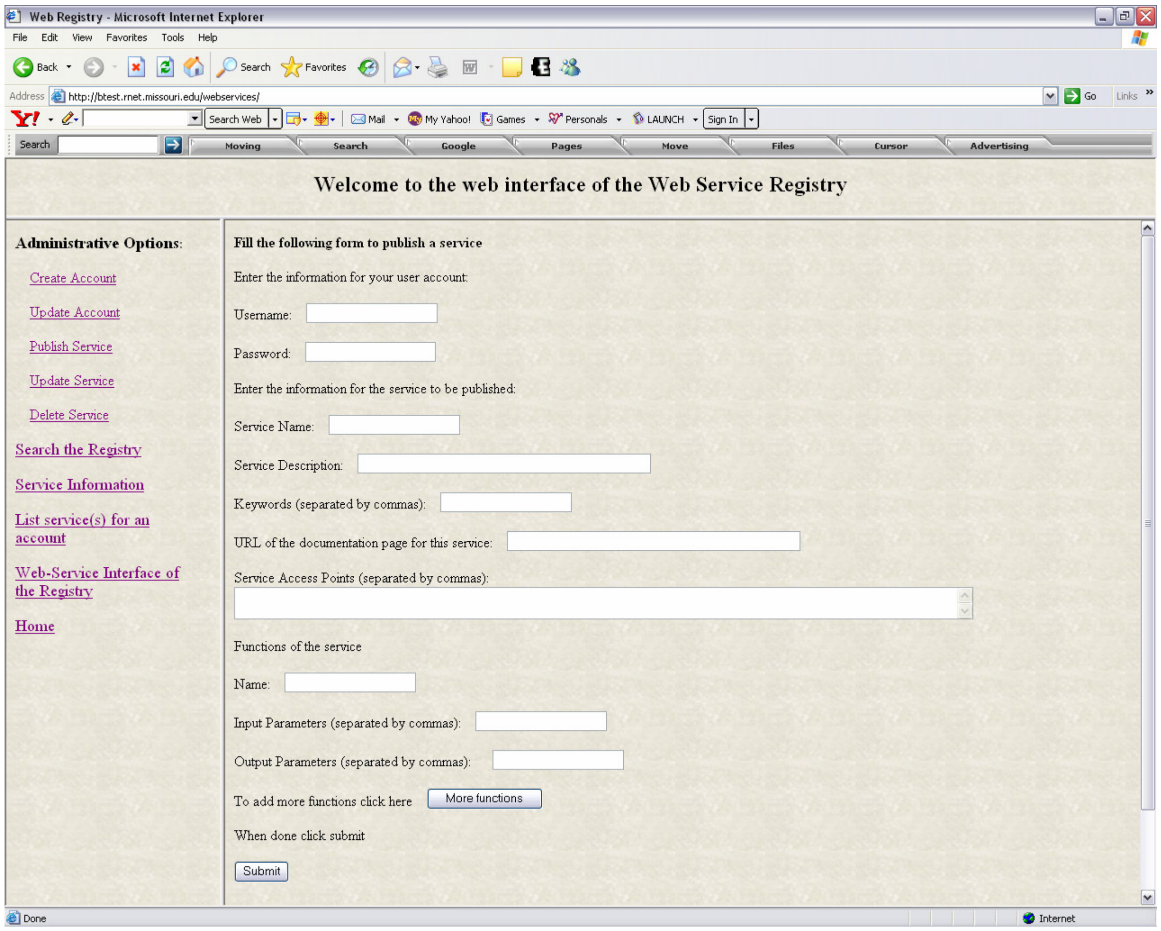
☐ Phone Number:

Submit

Figure 5.7: HTML Form to Update a Web Service Registry Account

5.1.1.2 Publish and Update a Web Service

The concept of a Web Service Registry originated because of the need to store information about web services. Hence, publishing a web service is an important operation executed by the Admin interface. The information stored in the WSR for a web service has to be comprehensive enough for the users to search the WSR for a particular web service. The HTML form displayed by following the link 'Publish Service' under the administrative options from the main page of the WSR is shown in Figure 5.8. Only users that have created a WSR account can publish a web service. The user has to provide the username and the password along with the web service information in order to be authenticated and to publish a web service with the WSR. The web service information to be entered is categorized into searchable and non-searchable fields. Searchable fields are those fields whose data is used to search a web service in the WSR. Non-searchable fields are used only to display information about a web service. The URL of the documentation page of a web service and the access points (addresses) of a web service are the non-searchable fields. The access point(s) of a web service are the URL(s), where the service can be accessed. The user can provide more than one access point, if the web service is running at multiple access points.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Games Personals LAUNCH Sign In
Search Moving Search Google Pages Move Files Cursor Advertising
Done
Internet

Welcome to the web interface of the Web Service Registry

Administrative Options:

- Create Account
- Update Account
- Publish Service
- Update Service
- Delete Service
- Search the Registry
- Service Information
- List service(s) for an account
- Web-Service Interface of the Registry
- Home

Fill the following form to publish a service

Enter the information for your user account:

Username:

Password:

Enter the information for the service to be published:

Service Name:

Service Description:

Keywords (separated by commas):

URL of the documentation page for this service:

Service Access Points (separated by commas):

Functions of the service

Name:

Input Parameters (separated by commas):

Output Parameters (separated by commas):

To add more functions click here [More functions]

When done click submit

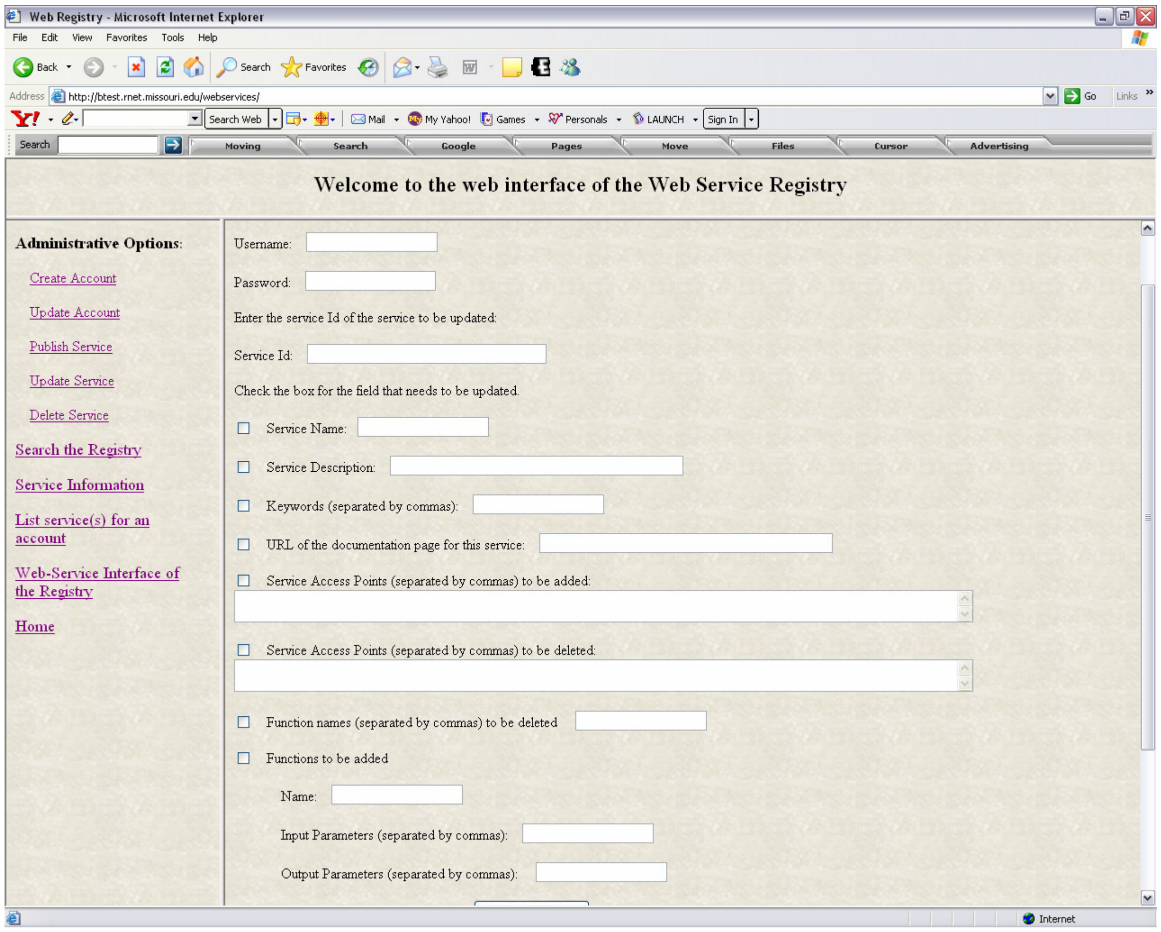
[Submit]

Figure 5.8: HTML Form to Publish a Web Service with the Web Service Registry

The searchable fields are the service name, the service description, the service key-words and the functions implemented by the service. The values for these fields are entered and stored in the Oracle database as text strings. The values entered for these fields are very crucial as they directly impact the search history of a web service. The values have to be relevant to the functionality of the web service. The description of the web service should have appropriate terminology; the key-words should have relevant and necessary words; and the functions should have appropriate names relevant to their tasks. The input and output parameters of each function are also used for the search operation. If a function has more than one input or output parameter, they can be entered separated by commas. Since a web service is quite likely to implement more than one function, provision is made to enter details for more functions. Once the information is entered and submitted, a program called CreateService is called on the remote server.

The program on the server parses the information sent by the HTML form and does an authentication check to see if an account exists for the supplied username and password. If the account does not exist or the password is incorrect, the user is sent an appropriate message indicating failure to authenticate. If a valid account exists, a universally unique identifier (UUID) is generated for the web service. The UUID is generated using the current timestamp, the server (machine) identifier where the program is running and the encoded string of the web service name. The parameters used to generate service UUID ensure that it is unique. The program forms am update message to insert the new service information along-with the service UUID and the UUID associated with the user account. The update message is sent to the Listener(s) to insert the data for the new web service in the Oracle database instance(s). Upon successful execution the program displays the UUID assigned to the web service to the user.

The information provided when the web service is published is important to the search history of the web service. Over time, the information might need to be updated to reflect changes made to the web service, such as deletion of some functions, addition of functions, change of input and output parameters of the functions and other modifications. The service information in the WSR must be kept up-to-date. The information for a web service can be modified following the ‘Update Service’ link under the administrative options. As shown in Figure 5.9, the form to update web service information is like the form used to publish a web service. The user needs to have the UUID of the web service he intends to update. The user can get the UUID(s) of the web service(s) published under his account by following the link ‘List services published for a given WSR account’ as explained in Section 5.1.2.



Welcome to the web interface of the Web Service Registry

Administrative Options:

- Create Account
- Update Account
- Publish Service
- Update Service
- Delete Service

Search the Registry

Service Information

List service(s) for an account

Web-Service Interface of the Registry

Home

Username:

Password:

Enter the service Id of the service to be updated:

Service Id:

Check the box for the field that needs to be updated.

☐ Service Name:

☐ Service Description:

☐ Keywords (separated by commas):

☐ URL of the documentation page for this service:

☐ Service Access Points (separated by commas) to be added:

☐ Service Access Points (separated by commas) to be deleted:

☐ Function names (separated by commas) to be deleted

☐ Functions to be added

Name:

Input Parameters (separated by commas):

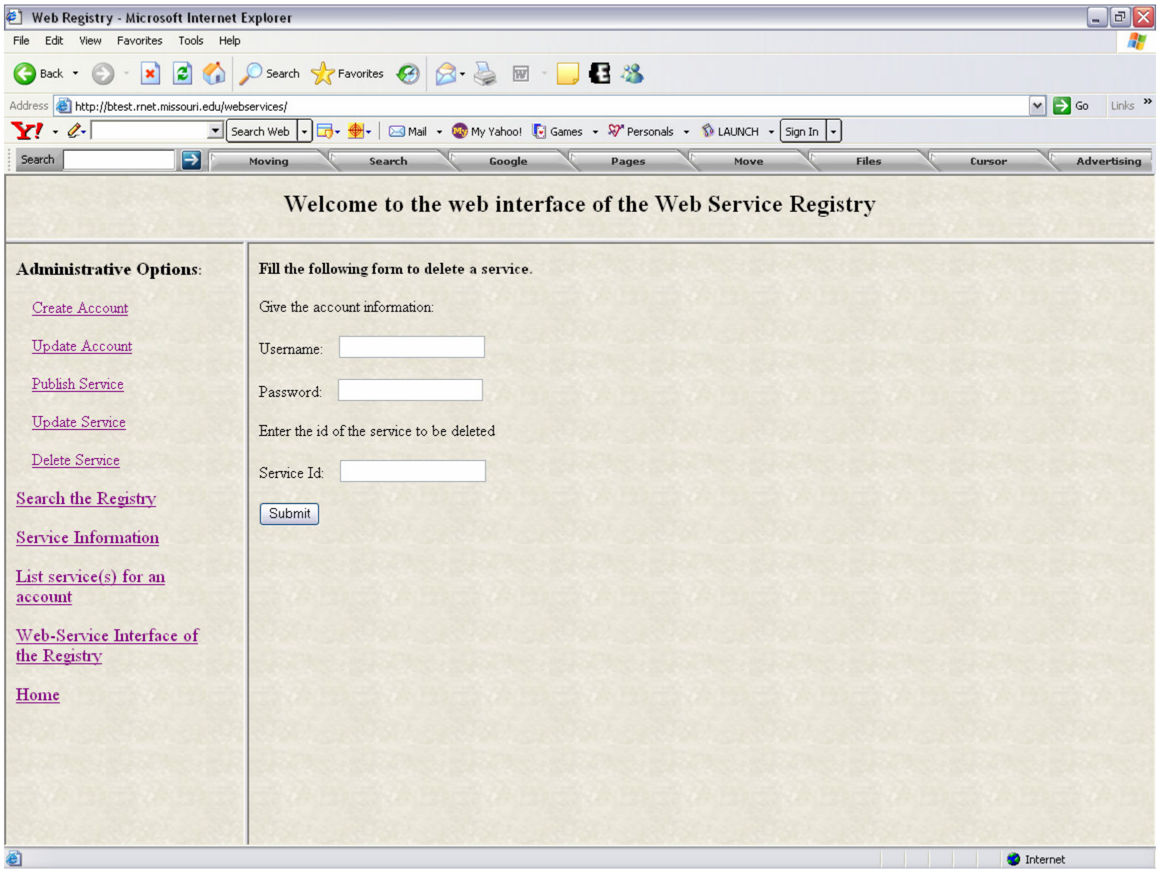
Output Parameters (separated by commas):

Figure 5.9: HTML Form to Update a Web Service in the Web Service Registry

Apart from the fields found on the HTML from to publish a web service in Figure 5.8, the new fields on the form to update a web service are the fields for the UUID of the web service and the check boxes to indicate the fields that need to be updated. The form calls the remote program UpdateService on the server and passes the information entered by the user. The user is authenticated and authorized to see if he is the owner of the service. After authentication and authorization, the program sends an update request on the multicast address to the Listener(s), which updates the service information in the Oracle database. The user is displayed a message indicating whether or not the web service information was updated.

5.1.1.3 Delete a Web Service

A web service, if not meant to be used or is not functional, should be removed from the WSR. The owner of the service has the ability to remove a web service from the WSR by using the link 'Delete Service' listed under the administrative options. The form is shown in Figure 5.10. The user is required to enter the username, the password and the UUID of the service he wants to remove from the WSR. When the form is submitted, it calls the program DeleteService on the remote server and passes the information entered by the user. The program after doing the authentication and authorization checks sends an update request to the Listener(s) to remove all the information of that service from the Oracle database. A message is displayed to the user indicating whether or not the service was successfully deleted from the WSR.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Games Personals LAUNCH Sign In
Search Moving Search Google Pages Move Files Cursor Advertising
Internet

Welcome to the web interface of the Web Service Registry

Administrative Options:

- Create Account
- Update Account
- Publish Service
- Update Service
- Delete Service
- Search the Registry
- Service Information
- List service(s) for an account
- Web-Service Interface of the Registry
- Home

Fill the following form to delete a service.

Give the account information:

Username:

Password:

Enter the id of the service to be deleted

Service Id:

Submit

Figure 5.10: HTML Form to Delete a Web Service from the Web Service Registry

The Admin operations give the user the ability to publish and update information in the WSR whereas the Inquire operations give the user the ability to search and retrieve information from the WSR. The Inquire operations supported by the Inquire interface of the CGI interface of the WSR are discussed below.

5.1.2 Inquire Interface

The Inquire operations are related to searching the Web Service Registry for the data requested by a user based on some search criteria. Based upon the nature of the search, the search for the data can be categorized into simple-search and complex-search. The search for data using values that exist as separate individual fields in the database is called a simple-search. For example, a web service unique identifier (UUID), or an account UUID exist as separate individual fields in the database and thus, their value is easily comparable to the value (string) entered by the user.

On the contrary, the search for data using values that do not exist as separate individual fields in the database but are present as a part of another value (string) stored in the database is termed as complex-search. The complex-search is also termed as text-search as it searches for words that may be present as a part of a long text field. For example, searching the database for a web service based on some key-words that may be present in the description field of a web service, or searching based on some parameters that may be present in the input or output parameter lists of the functions implemented by a web service is called a complex-search.

Simple-search uses the generic string comparison function to find the value (string) entered by the user that exists as a separate individual field in the database. Complex or text-search uses complex search functions that search for the different permutations that could possibly originate from a single value (string) entered by the user. For example, the key-word "phone" entered by the user should be able to match with words, "cell-phone", "tele-phone", "phone-booth", and much more that may be present in a long text string stored in the database. The Oracle database provides a search tool called "Intermedia Text" [13], which provides the text management and retrieval capabilities to allow complex or text searching. An index is created for every field in the database used for complex or text searching. Indexing keeps track of all the words present in a field. The script to create indexes is available in Appendix A. More information on the "Intermedia Text" can be found on the Oracle website [13]. The tool provides an interface to build SQL queries to perform complex or text search on the data stored in the Oracle database.

After having understood the two types of search (simple and complex or text) used by the Inquire operations of the WSR, the various operations provided by the Inquire interface are discussed below. The source code of the remote programs that perform the Inquire operations is available in Appendix C. The generalized execution of the Inquire operations is shown in Figure 5.11. The execution of an Inquire operation is similar to that of an Admin operation except for the fact that the remote programs do not perform write operations to the database.

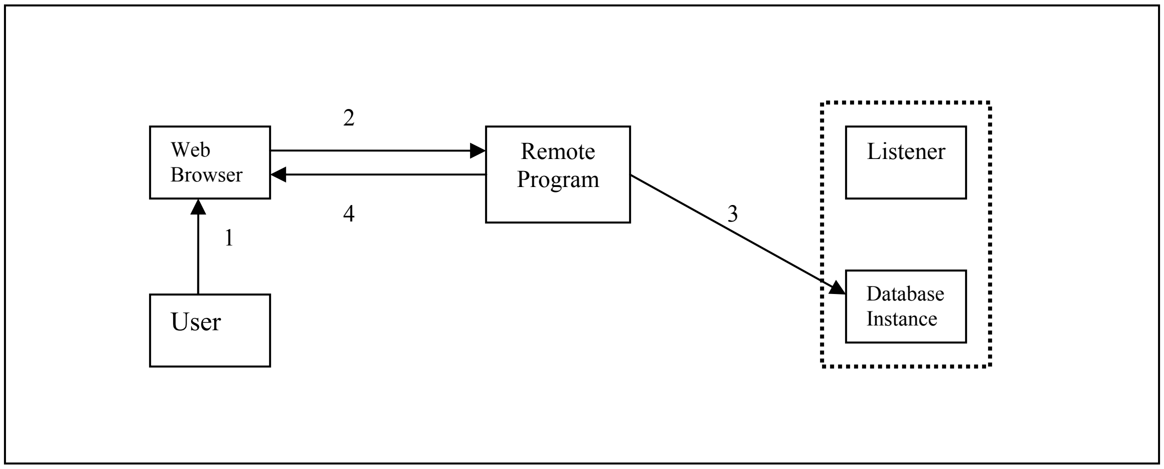
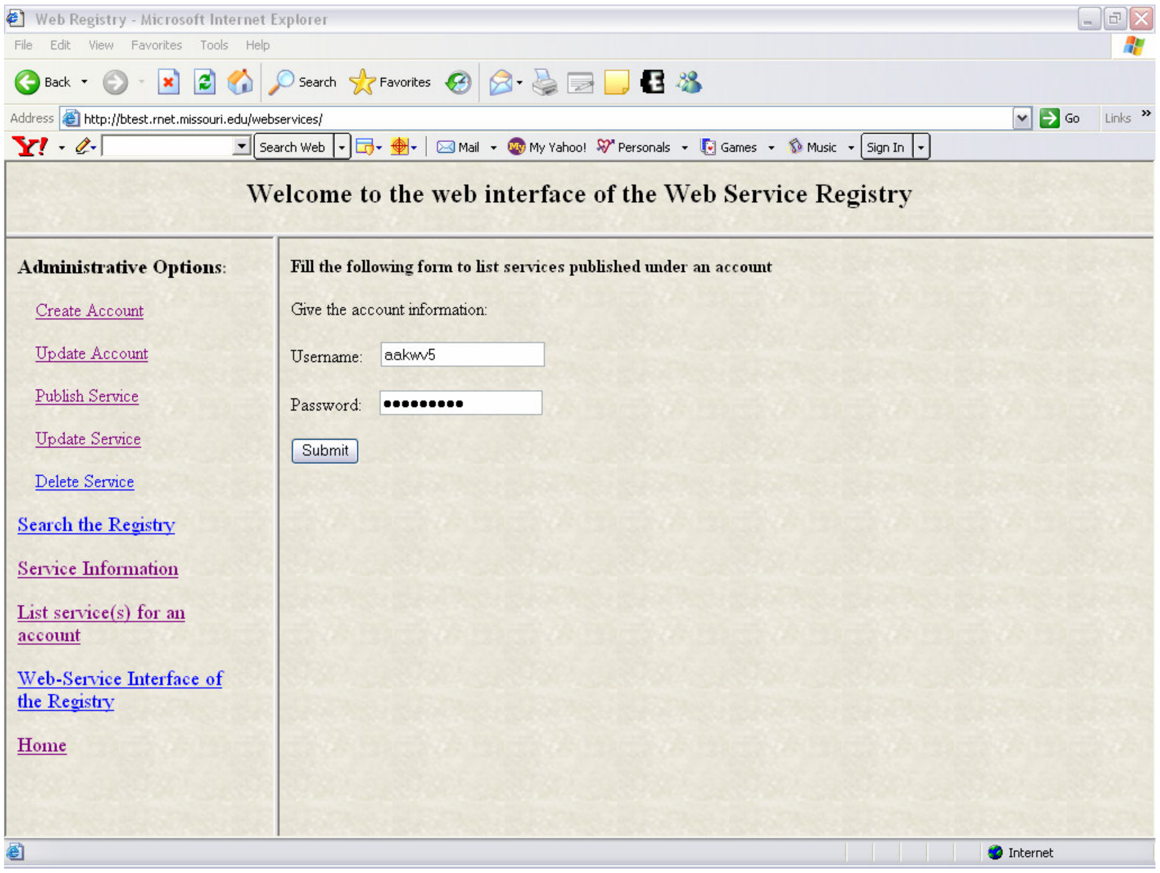


Figure 5.11: Execution of an Inquire Operation on the CGI Interface of the Web Service Registry

5.1.2.1 Retrieve Information of a Web Service

The user is allowed to publish as many web services he wants with the WSR. After a web service has been published, the user might want to update the information to keep the information of the web service up-to-date in the WSR. In order to update the information the web service unique identifier (UUID) assigned to the service when it was published, needs to be known. The service UUIDs of all the web services published by a user is made available using the ‘List service(s) for an account’ link on the main-page of the WSR. The form displayed to list the web services published by a user is shown in Figure 5.12.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Personals Games Music Sign In
Internet

Welcome to the web interface of the Web Service Registry

Administrative Options:

Create Account
Update Account
Publish Service
Update Service
Delete Service
Search the Registry
Service Information
List service(s) for an account
Web-Service Interface of the Registry
Home

Fill the following form to list services published under an account

Give the account information:

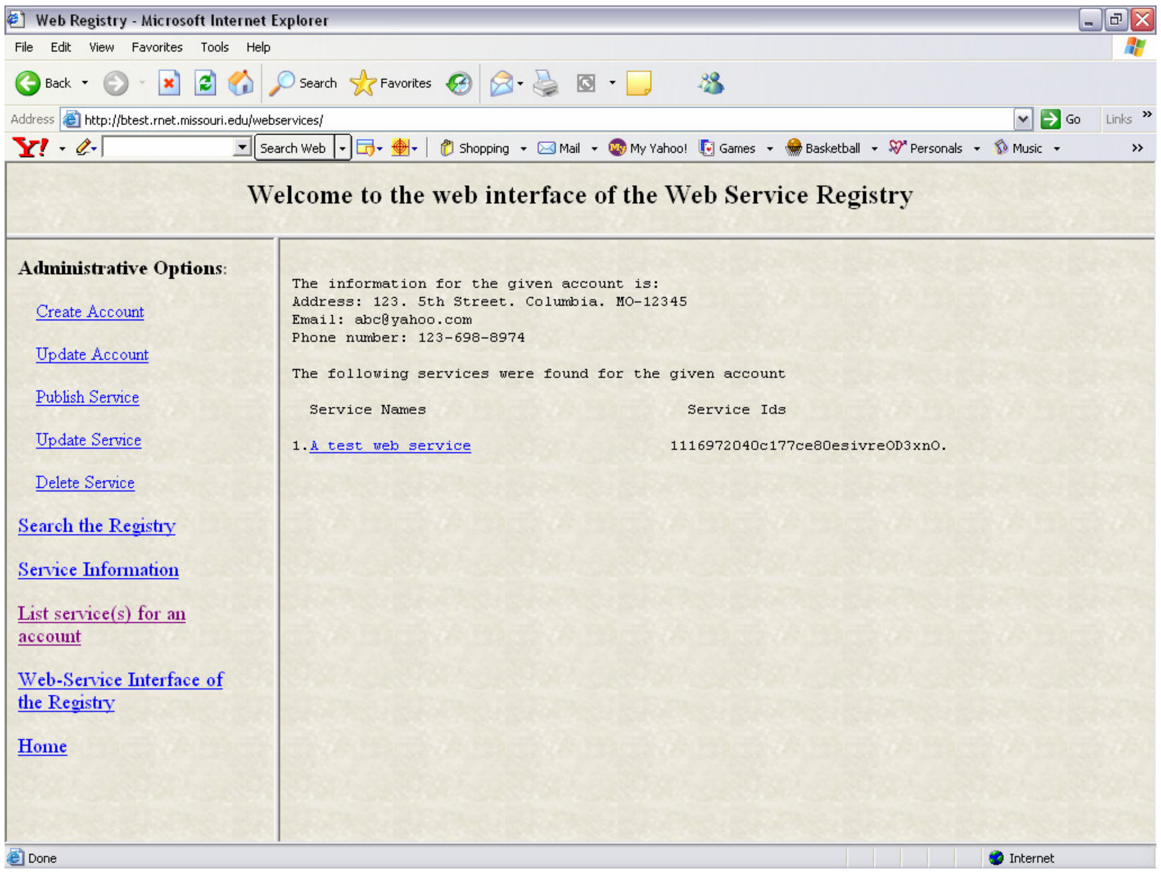
Username: aakwv5

Password: ●●●●●●●●●●

Submit

Figure 5.12: HTML Form to List Web Services Published by a User

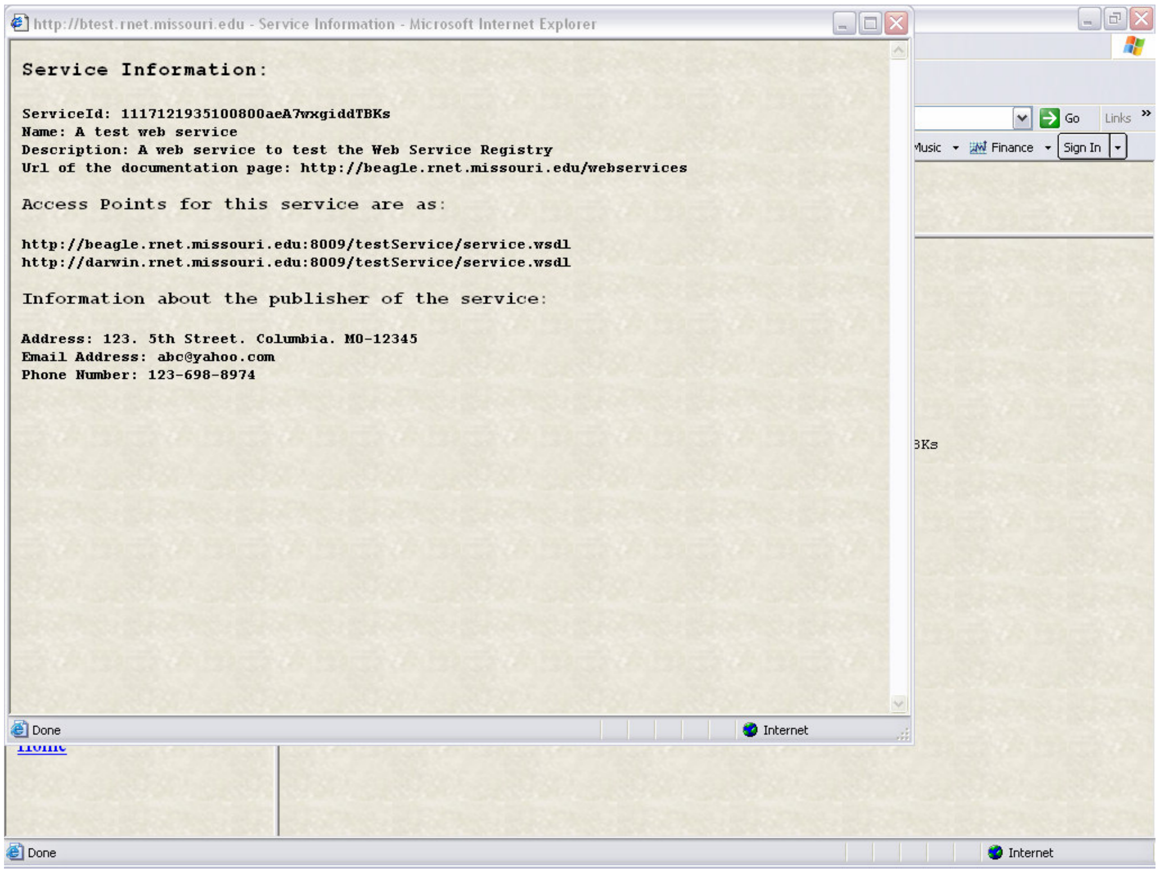
Users are required to enter their username and password for authentication. The form passes the username and the password to the remote program ServiceByAccount on the server. The program authenticates the user and retrieves the service UUIDs of the web services published by the user by remotely connecting to one of the Oracle database instance(s). The remote program returns the UUIDs of the services published by the user along-with with the name of the web services. Figure 5.13 shows the names and the UUIDs of the web services published by a given user.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://btest.rnet.missouri.edu/webservices/
Welcome to the web interface of the Web Service Registry
Administrative Options:
Create Account
Update Account
Publish Service
Update Service
Delete Service
Search the Registry
Service Information
List service(s) for an account
Web-Service Interface of the Registry
Home
Done
Internet

```
The information for the given account is:
Address: 123. 5th Street. Columbia. MO-12345
Email: abc@yahoo.com
Phone number: 123-698-8974

The following services were found for the given account

  Service Names                              Service Ids

1.A test web service                       1116972040c177ce80esivre0D3xn0.
```

Figure 5.13: Web Services Published by a User with the Web Service Registry

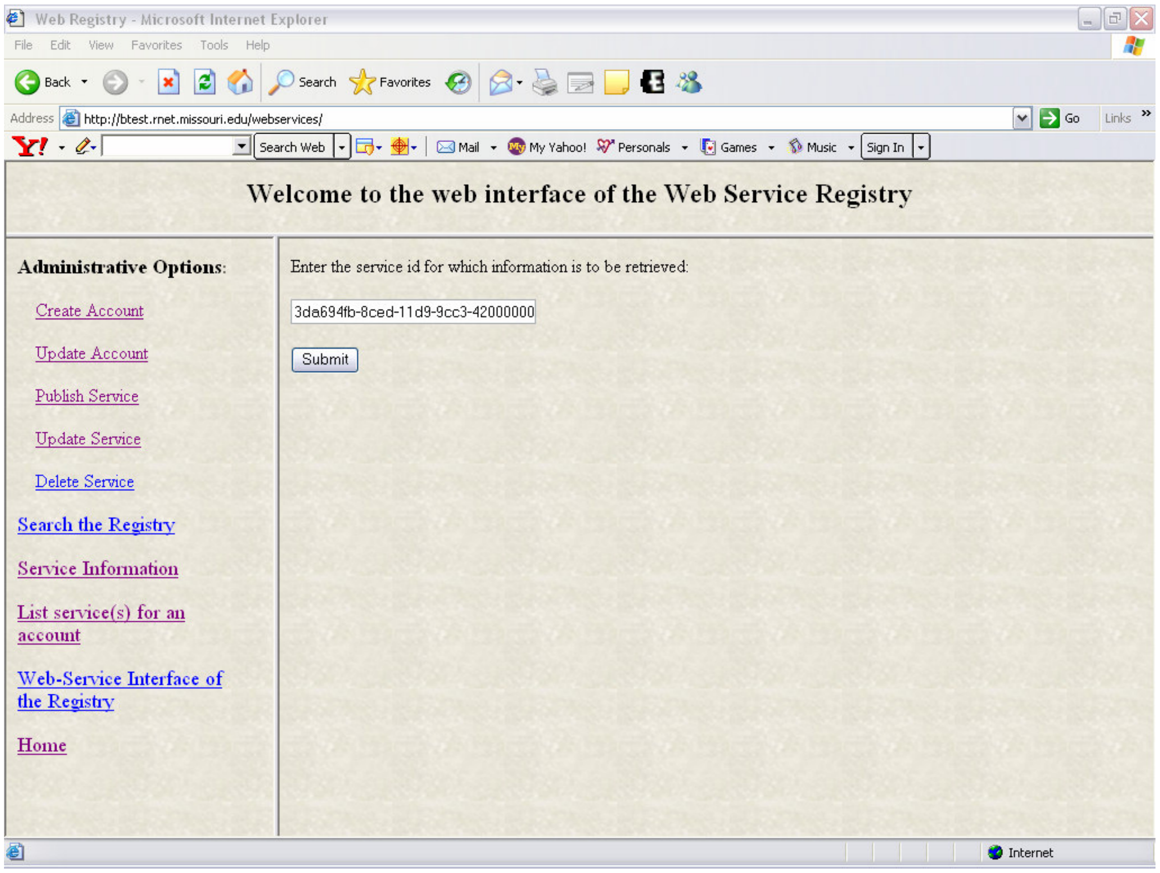
As visible in Figure 5.13, the names of the web services are displayed in the form of links. The link retrieves information for that web service from the WSR. When the user clicks on the link, a program called GetServiceById is called on the remote server. The program accepts the UUID of the web service for which information is to be retrieved. No authentication is done by the program as the information about a web service is accessible to anybody using the WSR. The program retrieves information about that web service by remotely connecting to a Oracle database instance. The information returned from the program is displayed to the user in a new window as shown in Figure 5.14.


http://btest.rnet.missouri.edu - Service Information - Microsoft Internet Explorer
Done
Internet
Go
Links
Music
Finance
Sign In
Home
3Ks
Done
Internet

```
Service Information:

ServiceId: 1117121935100800aeA7wxgiddTBKs
Name: A test web service
Description: A web service to test the Web Service Registry
Url of the documentation page: http://beagle.rnet.missouri.edu/webservices

Access Points for this service are as:

http://beagle.rnet.missouri.edu:8009/testService/service.wsdl
http://darwin.rnet.missouri.edu:8009/testService/service.wsdl

Information about the publisher of the service:

Address: 123. 5th Street. Columbia. MO-12345
Email Address: abc@yahoo.com
Phone Number: 123-698-8974
```

Figure 5.14: Information Retrieved for a given Web Service from the Web Service Registry

Information about a web service can also be retrieved by following the ‘Service Information’ link from the main page of the WSR. The link provides the user with a HTML form, as shown in Figure 5.15. The user is required to enter the service UUID of the service, the user seeks information for. The form calls the same program GetServiceById on the remote server and passes it the UUID of the service. The program retrieves information from the database and shows it to the user as shown in Figure 5.14.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Personals Games Music Sign In
Internet

Welcome to the web interface of the Web Service Registry

Administrative Options:

- Create Account
- Update Account
- Publish Service
- Update Service
- Delete Service

Search the Registry

Service Information

List service(s) for an account

Web-Service Interface of the Registry

Home

Enter the service id for which information is to be retrieved:

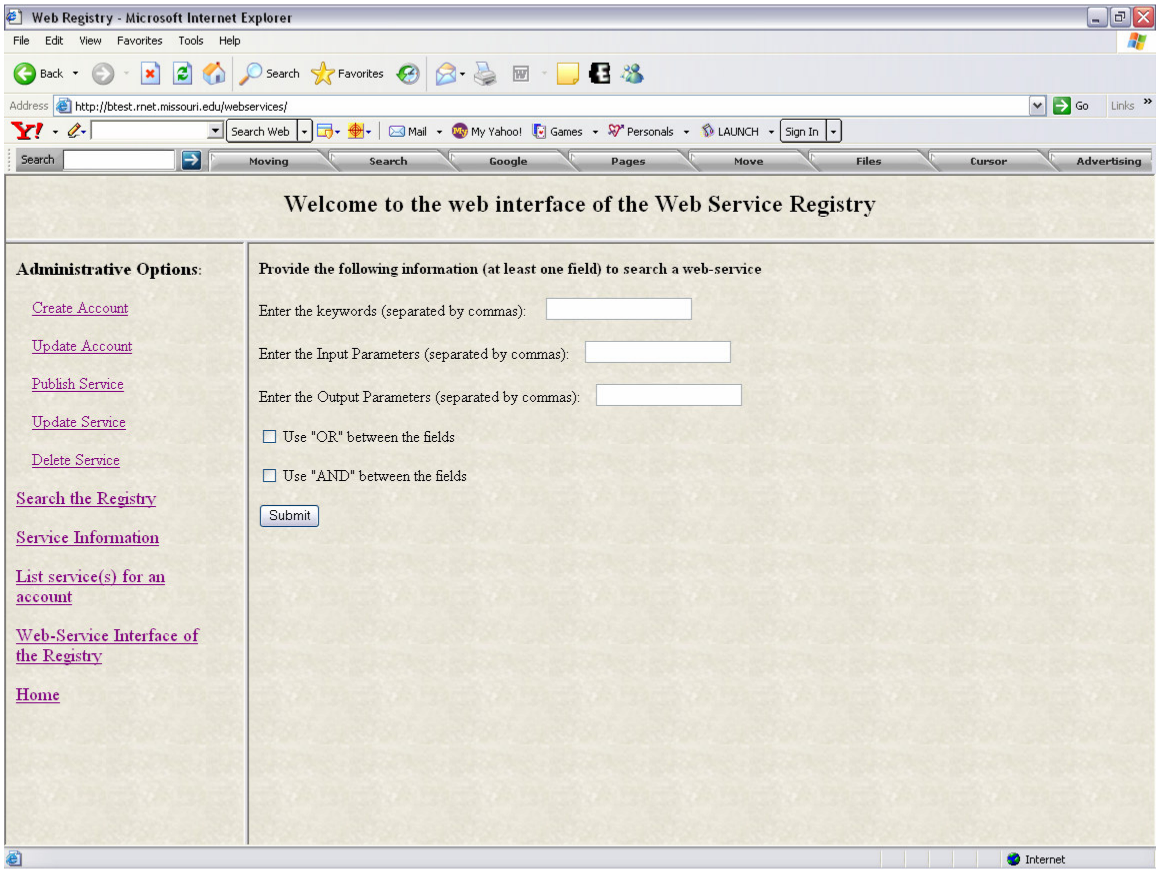
3da694fb-8ced-11d9-9cc3-42000000

Submit

Figure 5.15: HTML Form to Retrieve Information about a Web Service

5.1.2.2 Search the Web Service Registry for a Web Service

A user visits the Web Service Registry to search for web services that he is interested in utilizing to build his own applications. For example, the user might be interested in finding a web service published by others that accepts a credit card number and checks the validity of the number. The user could use this web service in his applications related to e-commerce. The link 'Search the Registry' provides the user with the capability to search the WSR for web service(s) based upon some search criteria entered by the user. The HTML form to perform search on the WSR is shown in Figure 5.16.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
http://btest.rnet.missouri.edu/webservices/

Welcome to the web interface of the Web Service Registry

Administrative Options:

Create Account

Update Account

Publish Service

Update Service

Delete Service

Search the Registry

Service Information

List service(s) for an account

Web-Service Interface of the Registry

Home

Provide the following information (at least one field) to search a web-service

Enter the keywords (separated by commas):

Enter the Input Parameters (separated by commas):

Enter the Output Parameters (separated by commas):

☐ Use "OR" between the fields

☐ Use "AND" between the fields

Submit

Figure 5.16: HTML Form to Search the Web Service Registry for Web Services

The fields to be entered include keywords that might be present in the name, description and/or the keywords of the web service published with the WSR. For example, the user might enter ‘credit-card, validation’ in the keywords field. The user can also enter the input parameters and/or the output parameters of the functions he is interested in and might be implemented by the web service published with the WSR. For example, he might enter ‘number’ in the input parameters field and “valid” in the output parameters field. At-least one field on the search form has to be filled to do the search. After the information is entered, the form calls the SearchServices program on the remote server. The program forms a query to search the database for the web services that match the search criteria entered by the user. The keywords entered by the user are searched in the name, the keywords and the description fields of all the web services published with the WSR. The input and the output parameters are searched in the input and the output parameter lists of the functions of all the web services. The text search is done using the tool “Intermedia Text” [13] such that most of the permutations originating from the words entered by the user are considered. The web services that match the search criteria are displayed to the user.

For example, the user looking for a credit-card validation web service enters “credit-card, validation” in the keywords box and enters “number” in the input parameters box. The user has the option of using “OR” or “AND” between the fields on the search form. If the user chooses the “OR” option, then the web services that have the words “credit-card” or “validation” in the name, the keywords or the description of the web service OR the web services that have a function with the input parameter “number”, are displayed.

If the user chooses the “AND” option, then only those web services with the words “credit-card” or “validation” in the name, the keywords or the description of the web service AND also have a function with the input parameter “number”, are displayed. This example of credit-card validation is merely for explanatory purposes and not an application web service implemented as part of this thesis.

As mentioned earlier, the fields that are searched in the Oracle database need to be indexed and the indexes must be rebuilt whenever a field is updated. The fields that store information about the web service are updated whenever a web service is added or removed from the WSR. The Listener that receives the request message to update the database rebuilds the indexes whenever data gets updated in those fields. The Oracle Reference Guide [13] contains information on creating and rebuilding the indexes.

5.1.2.3 Retrieve Information of the Web Service Interface of the Web Service Registry

As mentioned earlier, the WSR also implements a web service interface in addition to the CGI interface. The web service interface is provided to the WSR so that all the Admin and Inquire operations discussed earlier can also be accessed programmatically from within a web service client. The web service interface is discussed in detail in the following section.

The web service interface comprises of two web services. One web service provides the Admin interface and the other provides the Inquire interface. The web services are called WsrAdminSoapServer and WsrInquireSoapServer respectively. These web services are written in JAVA using GLUE [11]. The procedure to build web services using GLUE is discussed in "Web Services Building Blocks for Distributed Systems" [1].To access WsrAdminSoapServer and WsrInquireSoapServer web services from a web service client the user needs to know the access points (addresses) of the two web services. This information is made available by the link 'Web-Service Interface of the Registry' on the main page of the WSR. This link calls the program GetWebRegInfo on the remote server that retrieves information for the web services, WsrAdminSoapServer and WsrInquireSoapServer, and displays it to the user.

In Chapter 4 (Section 4.11) we discussed the performance evaluation of the procedures executed on the fault-tolerant system to start and administer a web service cluster. The following section discusses the performance evaluation of the operations of the WSR executed from the CGI interface of the WSR.

5.1.3 Performance Evaluation

The various operations available on the web interface (CGI interface) of the WSR were executed several times for a period of seven days to gather statistics on the execution time of each operation. The remote programs of the web interface of the WSR are deployed on a Compaq Tru64 UNIX V5.1B, 4 processor system. As mentioned earlier, the operations of the WSR are categorized as Admin and Inquire operations. The Admin operations update the database and the Inquire operations retrieve information from the database and make it available to the user of the WSR. The components of the fault-tolerant system that participate in executing the Admin operations are the remote programs that receive the request from the web interface, and the

Listener that receives the multicast update requests from the remote programs. For an Inquire operation, the remote program connects to the remote database instance and retrieves the necessary information.

The various Admin operations supported by the WSR are the update account, publish service, update service, and delete service. Table 5.1 shows the statistics of the CPU utilization and the execution times of the various Admin operations. Table 5.1 shows the minimum time taken, the maximum time taken and the average time taken to execute the operations. As mentioned earlier, the operations were executed daily for a span of seven days. The operations were run three times daily at different times during the day. The average is computed over 21 runs for each of the operations. The execution times of these operations include the time it takes to update the database through a multicast update message to the Listener(s).

Operations	Database transaction	Average (CPU : Execution)	Minimum (CPU : Execution)	Maximum (CPU : Execution)
Update Account	Write	0.023 sec : 2.9 sec	0.016 sec : 1 sec	0.05 sec : 6 sec
Publish Service	Write	0.025 sec : 12 sec	0.016 sec : 8 sec	0.06 sec : 18 sec
Update Service	Write	0.027 sec : 9.5 sec	0.016 sec : 6 sec	0.033 sec : 15 sec
Delete Service	Write	0.023 sec : 11.8 sec	0.016 sec : 9 sec	0.033 sec : 15 sec

Table 5.1: Statistics of the Admin Operations of the CGI Interface of the Web Service Registry

Table 5.2 shows the time taken by the Admin operations to update the database through a multicast update message to the Listener. The time taken for doing an update on the database varies based upon the information needed to be updated in the database. Some of the tables in the database that store information about the web services are indexed. Indexing the tables helps to search the information contained in those tables while searching for a web service based on the search criteria provided by the user. Updating a table that has indexes associated with it takes longer than updating other tables. This is because the indexes have to be rebuilt to accommodate the updated information. Table 5.2 shows the statistics for multicast updates with and without the indexes rebuilt.

Multicast Update	Average (Execution)	Minimum (Execution)	Maximum (Execution)
Without index(s) rebuilt	2.10 sec	1 sec	3 sec
With index(s) rebuilt	9.5 sec	6 sec	13 sec

Table 5.2: Statistics of the Multicast Update for the Admin Operations

Among the Admin operations, there are some operations that require the indexes to be rebuilt in the database. As shown in Table 5.1, the Update Account operation executes faster compared to the other operations that require indexes to be rebuilt. From Table 5.1 and Table 5.2, it is evident that the execution time of an Admin operation is approximately equal to the time taken to do a multicast update.

The Inquire operations are read operations and hence execute faster compared to the Admin operations. Table 5.3 shows the CPU utilization and the execution time of the Inquire operations such as searching the WSR, retrieving information for a given WSR account and retrieving information for a given web service.

Operations	Database transaction	Average (CPU : Execution)	Minimum (CPU : Execution)	Maximum (CPU : Execution)
Search the WSR	Read	0.018 sec : 2.2 sec	0.016 sec : 1 sec	0.033 sec : 4 sec
Retrieve information for a WSR account	Read	0.017 sec : 1.9 sec	0.016 sec : 1 sec	0.033 sec : 5 sec
Retrieve information for a web service	Read	0.026 sec : 1.6 sec	0.016 sec : 1 sec	0.033 sec : 2 sec

Table 5.3: Statistics of the Inquire Operations of the CGI Interface of the Web Service Registry

As seen in Table 5.1 and Table 5.3, the CPU utilization of all the operations (Admin and Inquire) is approximately the same. This is because the remote programs that execute the operations use the CPU of the server (machine) to the same extent. After having looked at the web interface (CGI interface), the following section discusses the web service interface of the WSR that provides the user the ability to programmatically access the WSR.

5.2 Web Service Interface of the Web Service Registry

The web services, WsrAdminSoapServer and WsrInquireSoapServer, which comprise the web service interface, are started as web service clusters using the fault-tolerant system discussed in Chapter 3 and Chapter 4. The fault-tolerant system ensures that at-least one instance of each, WsrAdminSoapServer and WsrInquireSoapServer, is always available. Thus, the web service interface of the WSR is designed to be fault-tolerant. These web services, when started as a cluster using the fault-tolerant system, automatically get published with the WSR. The user has to visit the CGI interface of the WSR to get the access points (addresses) of WsrAdmiinSoapServer and WsrInquireSoapServer. Once the user has the access points of these web services, he can access the WSR programmatically using a web service client.

As discussed in Section 4.10, any web service instance started as a web service cluster using the fault-tolerant system communicates its failure to the Web Service Manager through WsmSoapServer. The WsmSoapServer is a web service that receives notification of a failure from the web service instances of other web service clusters. The WsmSoapServer forwards the notification of failure to the Web Service Manager. Since WsrAdminSoapServer and WsrInquireSoapServer are started by the fault-tolerant system as web service clusters, the instances of WsrAdminSoapServer and WsrInquireSoapServer notify the Web Service Manager of their failure through WsmSoapServer.

WsrAdminSoapServer and WsrIquireSoapServer have access to the Oracle database where the access point(s) of the WsmSoapServer are stored. The WsrAdminSoapServer and the

WsrInquireSoapServer periodically read the access point(s) of the WsmSoapServer from the Oracle database. When any instance of WsrAdminSoapServer or WsrInquireSoapServer is about to fail, it tries to communicate the failure to the Web Service Manager through WsmSoapServer.

The operations supported by the web service interface, WsrAdminSoapServer and WsrInquireSoapServer, are similar to the Admin and Inquire operations supported by the CGI interface of the WSR. The Admin operations of the WSR help the users to publish a web service, create an account, and update WSR account and web service information. The Inquire operations help the users to search the WSR for web services and other operations such as retrieving information about a web service. The WsrAdminSoapServer and the WsrInquireSoapServer, which implement the Admin and the Inquire operations respectively, are discussed below.

5.2.1 WsrAdminSoapServer (Admin Interface)

The WsrAdminSoapServer is built in JAVA using the GLUE [11] Soap Factory. It implements the administrative operations as a set of functions that can be remotely called from a web service client. The functions implemented by WsrAdminSoapServer are createAcccount(), updateAccount(), createService(), updateService() and deleteService(). When WsrAdminSoapServer is started, it contacts the Listener(s) using multicast to get the address(s) of the database instance(s). The WsrAdminSoapServer remotely connects to the database instance to do a read operation, and sends an update request to update the database to the Listener(s) using multicast. Since the administrative operations involve transfer of confidential information, the WsrAdminSoapServer can be run using a HTTPS server. The information needed to start a web service using a HTTPS server is available in 'Web Services Building Blocks for Distributed Systems' [1].

The thesis implements a client program, WsrAdminSoapClient, to demonstrate the usage of WsrAdminSoapServer to perform the Admin operations programmatically. The source code of WsrAdminSoapClient and WsrAdminSoapServer can be found in Appendix C.

5.2.2 WsrInquireSoapServer (Inquire Interface)

WsrInquireSoapServer, like WsrAdminSoapServer, is built in JAVA using the GLUE [11] Soap Factory. It gets the address(s) of the database instance(s) by contacting the Listener(s) using multicast. It implements some of the operations of the CGI Inquire interface by implementing the functions getServiceAccessPoints(), findServices() and getServiceInfo(). These functions can be programmatically accessed by a web service client.

The getServiceInfo() function accepts the unique identifier (UUID) of a web service and returns information about the web service such as the service name, the service description, the access point(s), and the URL of the documentation page of the web service. This function is called when a client needs to know information about a given web service, for which he has its unique identifier (UUID).

The findServices() function corresponds to the ‘Search the Registry’ capability provided by the CGI Inquire Interface. The findServices() function serves the purpose of searching the WSR based on some search criteria, but it returns only the UUID(s) of the web service(s) that match the criteria. If a client is interested in finding more information about a web service, he can call the getServiceInfo() function and pass it the unique identifier (UUID) of the web service he is interested.

The getServiceAccessPoints() function retrieves the access point(s) of a given web service when provided with the unique identifier (UUID) of the web service. The function accepts the UUID of a web service and returns the access point(s) of that web service. If the web service has more than one access point; the function returns the access points in a different order every time the function is called.

The getServiceAccessPoints() function is used to provide transparency to a client utilizing a web service published with the WSR. For example, the web service ‘simpleService’ is published with the WSR and is run on more than one server (machine). The client program retrieves the access points of the ‘simpleService’ from the WSR using the getServiceAccessPoints() function. The client module contacts the ‘simpleService’ instance at the first access point returned by the WSR. If the ‘simpleService’ instance is not available at that access point, the client module can contact the ‘simpleService’ instance at other access points returned by the WSR. Thus, the failure of the instance at the first access point does not hinder the execution of the client program as it can contact the ‘simpleService’ at other access points. If all the access points are not reachable, then the client module can call the getServiceAccessPoints() again to get new access point(s) of the ‘simpleService’. Since, the ‘simpleService’ is run as a web service cluster of instances monitored by the fault tolerant system, there is always one instance of the service running and hence, the client module will find an instance to communicate with. This is true provided the network is not broken between the client and the web service instance(s). The order in which the access points of a web service are returned to the client by the function is randomly chosen so that a particular web service instance is not over-loaded with requests.

The web service instances of any web service cluster started by the fault-tolerant system get the access point(s) of the WsmSoapServer from the WSR. This is achieved by calling the function getServiceAccessPoints() and passing it the service UUID of the WsmSoapServer. The web service instances communicate their failure to the WsmSoapServer which in turn notifies the Web Service Manager.

The thesis implements a client program, WsrInquireSoapClient, to demonstrate the usage of WsrInquireSoapServer to perform the Inquire operations programmatically. The source code of WsrInquireSoapClient and WsrInquireSoapServer can be found in Appendix C. The last section of this chapter demonstrates the usage of the WSR by building an application web service, having it registered with the WSR and building a client application that uses that web service.

5.3 An Application Web Service and a Web Service Client

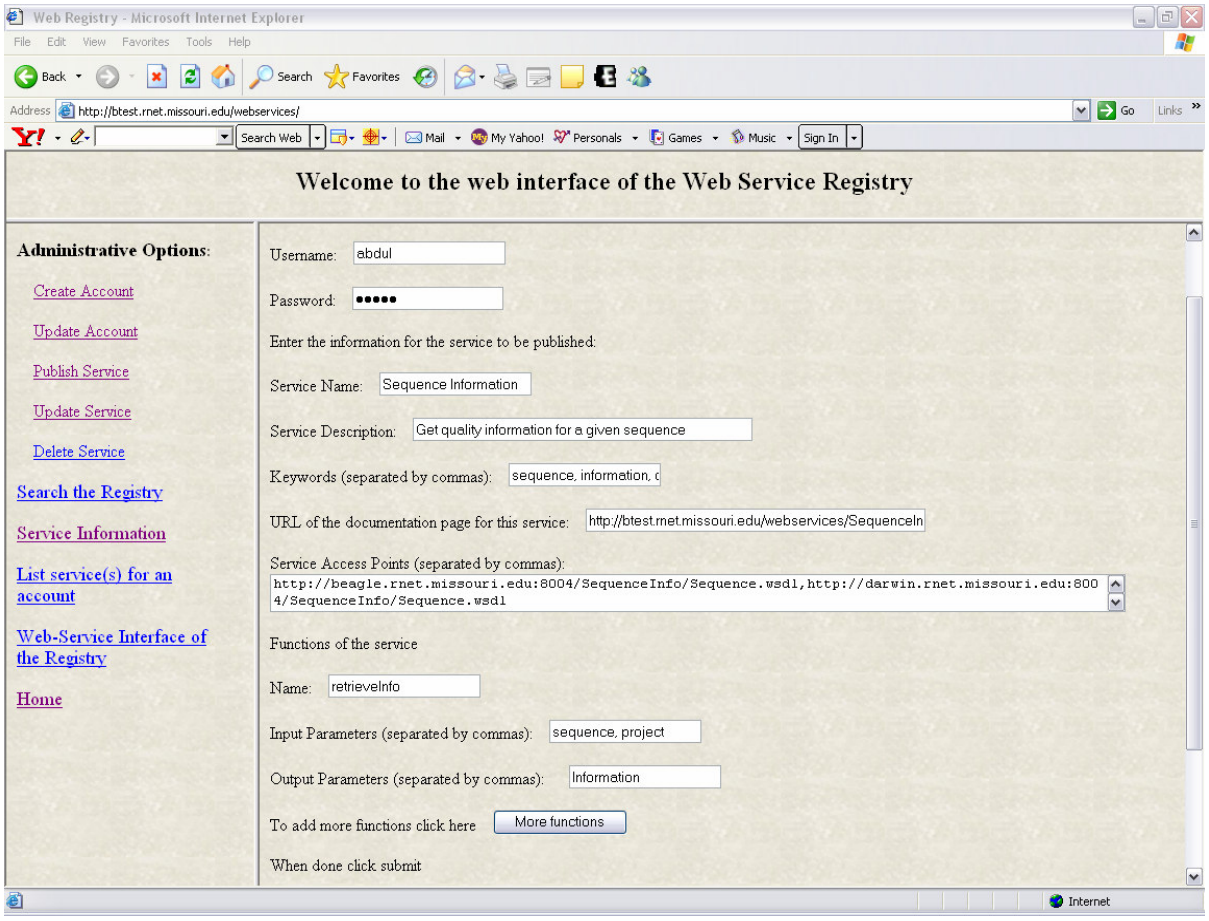
There are several on-going genomic projects at the University of Missouri-Columbia that provide information about genomic sequences, being studied at the University, to the community of researchers all over the world. Each project runs a web site, where the information is available in the form of web pages that can be accessed using a web browser. The projects can be accessed at http://genome.rnet.missouri.edu.

With the help of web services, the same information can be made available programmatically. The researchers can call web services from their programs to retrieve data and use the data in their programs. Every genomic sequence in the genomic projects is run through a process that determines the quality of the sequence and the information is stored in a genomic database. The application web service built for this thesis is a small effort to provide a web service interface to programmatically retrieve information about the quality of a sequence.

The web service is called 'SequenceInfo' and is built using the GLUE [11] Soap Factory. The source code of the web service is available in Appendix C. The web service implements a function called retrieveInfo() that accepts the name of a project and the name of a sequence as input parameters to the function. The function returns a JAVA object, called Information, which is a data structure used to pack information about the quality of a sequence. The Information object has many fields, each field carrying some information about the quality of the sequence. The structure of the object is available in Appendix C.

The prototype of the function retrieveInfo() implemented by 'SequenceInfo' looks like **public Information retrieveinfo(String project, String sequence);** In the prototype, *project* and *sequence* are the String data-type arguments received by the function; and *Information* is the java object (data structure) used to send information about a sequence to the caller. This function looks up the information for a sequence in a genomic database based on a project name and returns the information to the caller in the Information object.

After the web service is built and deployed, it has to be advertised for the potential clients to utilize the web service. Publishing the web service with the web WSR is the best way to have it advertised to the clients. Hence, this web service is registered with the WSR by providing relevant information for the description, key-words and input-output parameters fields, so that the clients can easily search for the web service in the WSR. The web service is published using the CGI interface of the WSR. The information entered on the form to publish the web service is shown in Figure 5.17.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Personals Games Music Sign In
Internet

Welcome to the web interface of the Web Service Registry

Administrative Options:

- Create Account
- Update Account
- Publish Service
- Update Service
- Delete Service
- Search the Registry
- Service Information
- List service(s) for an account
- Web-Service Interface of the Registry
- Home

Username: abdul

Password: •••••

Enter the information for the service to be published:

Service Name: Sequence Information

Service Description: Get quality information for a given sequence

Keywords (separated by commas): sequence, information, c

URL of the documentation page for this service: http://btest.rnet.missouri.edu/webservices/SequenceIn

Service Access Points (separated by commas):

```
http://beagle.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl,http://darwin.rnet.missouri.edu:800
4/SequenceInfo/Sequence.wsdl
```

Functions of the service

Name: retrieveInfo

Input Parameters (separated by commas): sequence, project

Output Parameters (separated by commas): Information

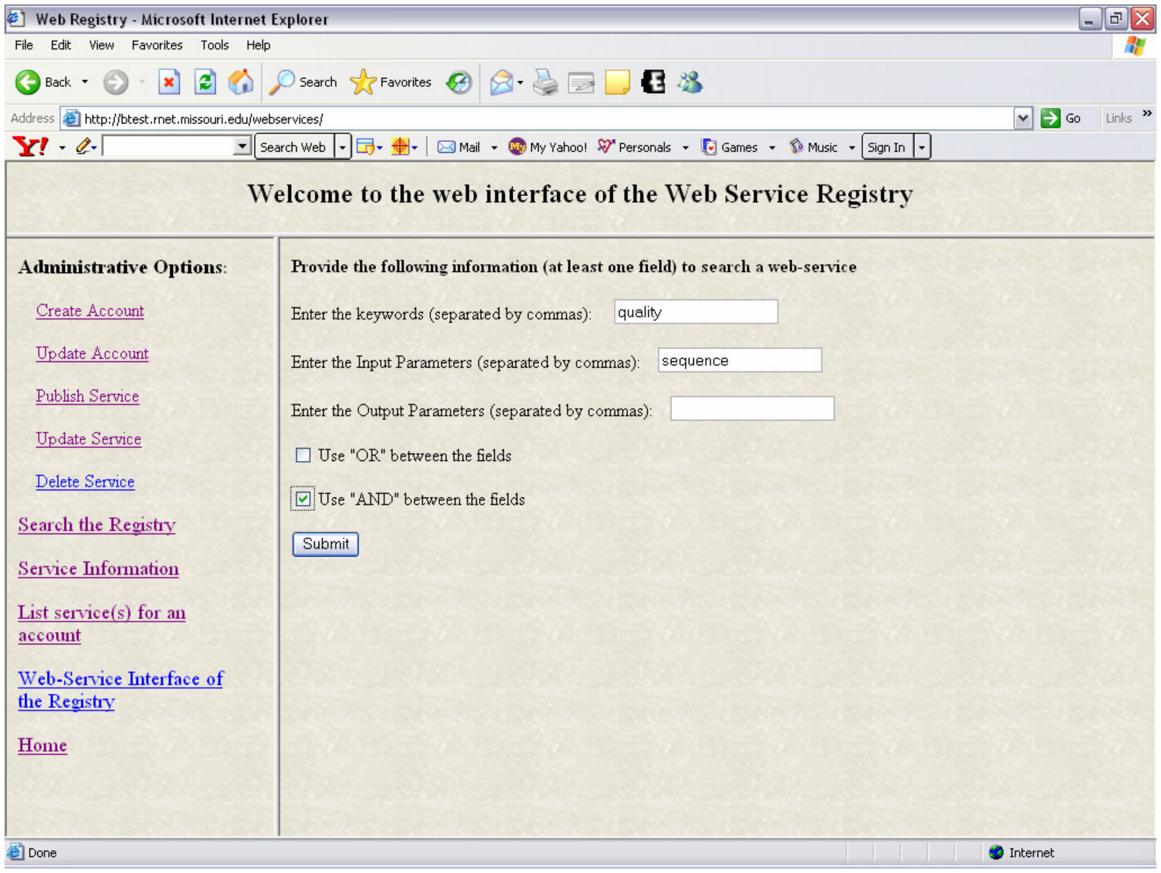
To add more functions click here [More functions]

When done click submit

Figure 5.17: Information Entered to Publish 'SequenceInfo' Web Service with the Web Service Registry

The clients can search the registry for this web service by entering the keywords, such as 'sequence', 'information', 'genome' and other keywords while searching the WSR, as shown in

Figure 5.18. The results obtained from the search are shown in Figure 5.19. Based upon the keywords entered, the search program was able to find the ‘SequenceInfo’ web service. The results show the name of the service and the service UUID.


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Personals Games Music Sign In

Welcome to the web interface of the Web Service Registry

Administrative Options:

Create Account

Update Account

Publish Service

Update Service

Delete Service

Search the Registry

Service Information

List service(s) for an account

Web-Service Interface of the Registry

Home

Provide the following information (at least one field) to search a web-service

Enter the keywords (separated by commas): quality

Enter the Input Parameters (separated by commas): sequence

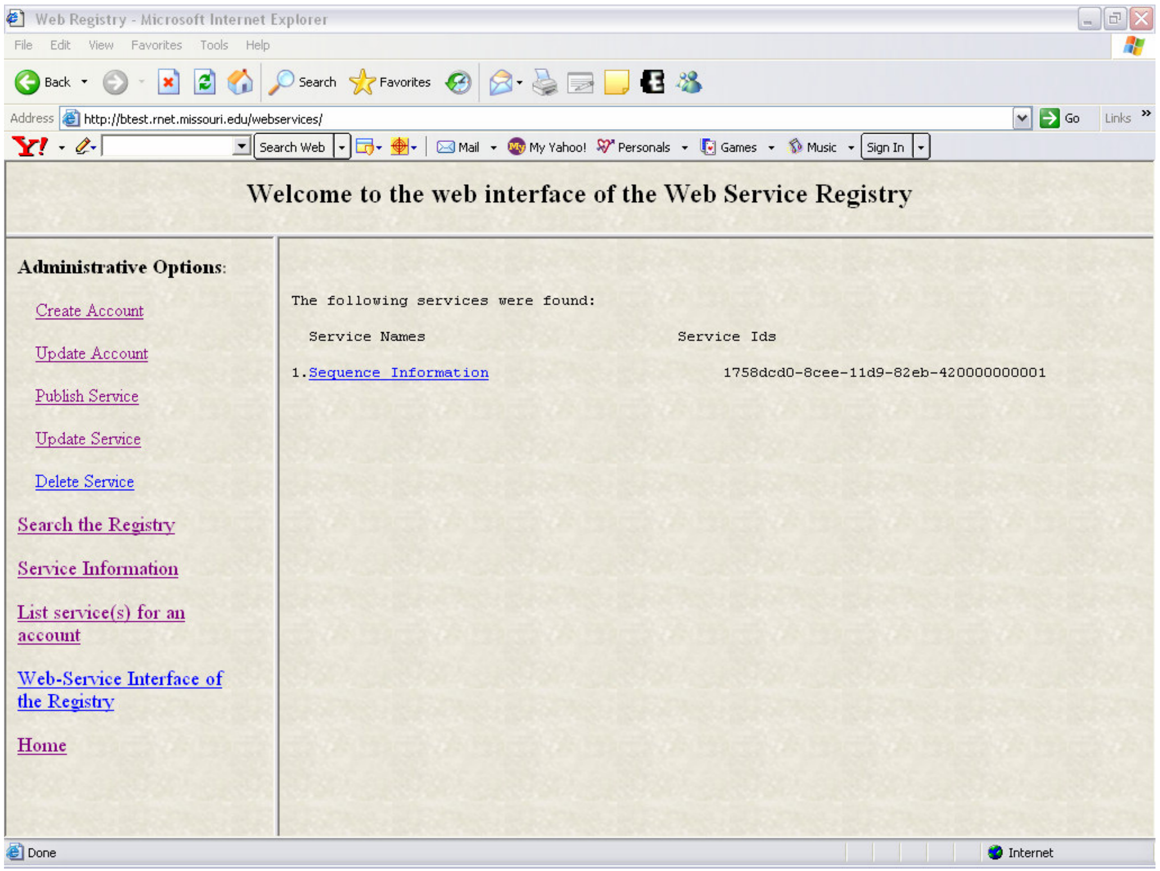
Enter the Output Parameters (separated by commas):

☐ Use "OR" between the fields

☑ Use "AND" between the fields

Submit

Figure 5.18: Searching for a Web Service on the Web Service Registry


Web Registry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://btest.rnet.missouri.edu/webservices/
Go Links
Search Web Mail My Yahoo! Personals Games Music Sign In
Done
Internet

Welcome to the web interface of the Web Service Registry

Administrative Options:

Create Account

Update Account

Publish Service

Update Service

Delete Service

Search the Registry

Service Information

List service(s) for an account

Web-Service Interface of the Registry

Home

```
The following services were found:

  Service Names                              Service Ids

1.Sequence Information                            1758dcd0-8cee-11d9-82eb-420000000001
```

Figure 5.19: Results Obtained from the Search on the Web Service Registry

Once the client knows the service unique identifier (UUID) of the web service, he can retrieve the access points of the web service programmatically by calling the WsrInquireSoapServer of the WSR. A client program SequenceInfoClient is written to call the web service SequenceInfo. The source code of the client program is available in Appendix C.

The process followed by SequenceInfoClient to call SequenceInfo is shown in Figure 5.20. The client calls the WsrInquireSoapServer of the WSR to get the access points of 'SequenceInfo' and passes to it the service unique identifier (UUID) of 'SequenceInfo' [step 1]. The WsrInquireSoapServer retrieves the access points of 'SequenceInfo' from the Oracle database [step 2] and sends them to the client [step3]. The client tries to contact the 'SequenceInfo' at any one of the access points returned by the WsrInquireSoapServer and calls the retrieveInfo() function passing it the project name and the sequence name [step 4]. The SequenceInfo returns the information about the sequence to the client [step 5].

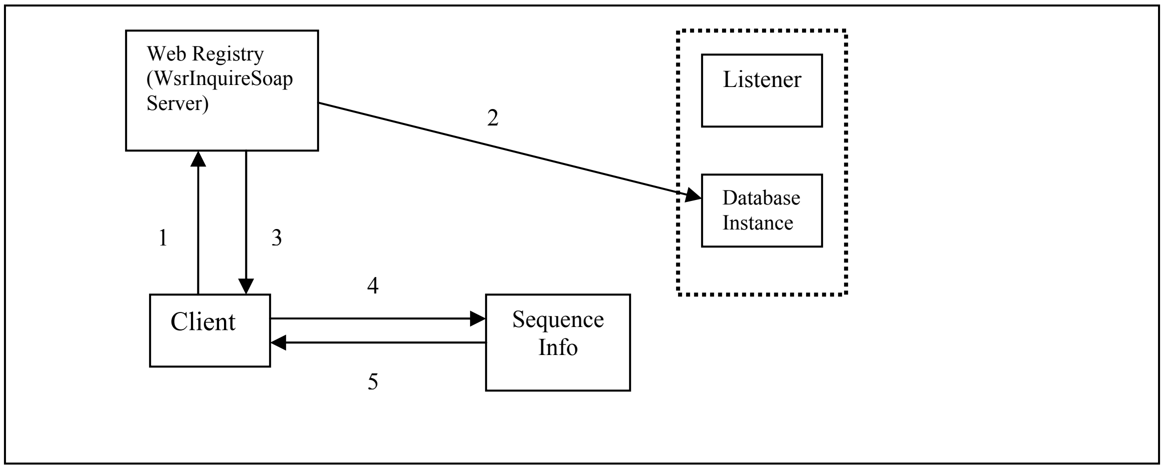
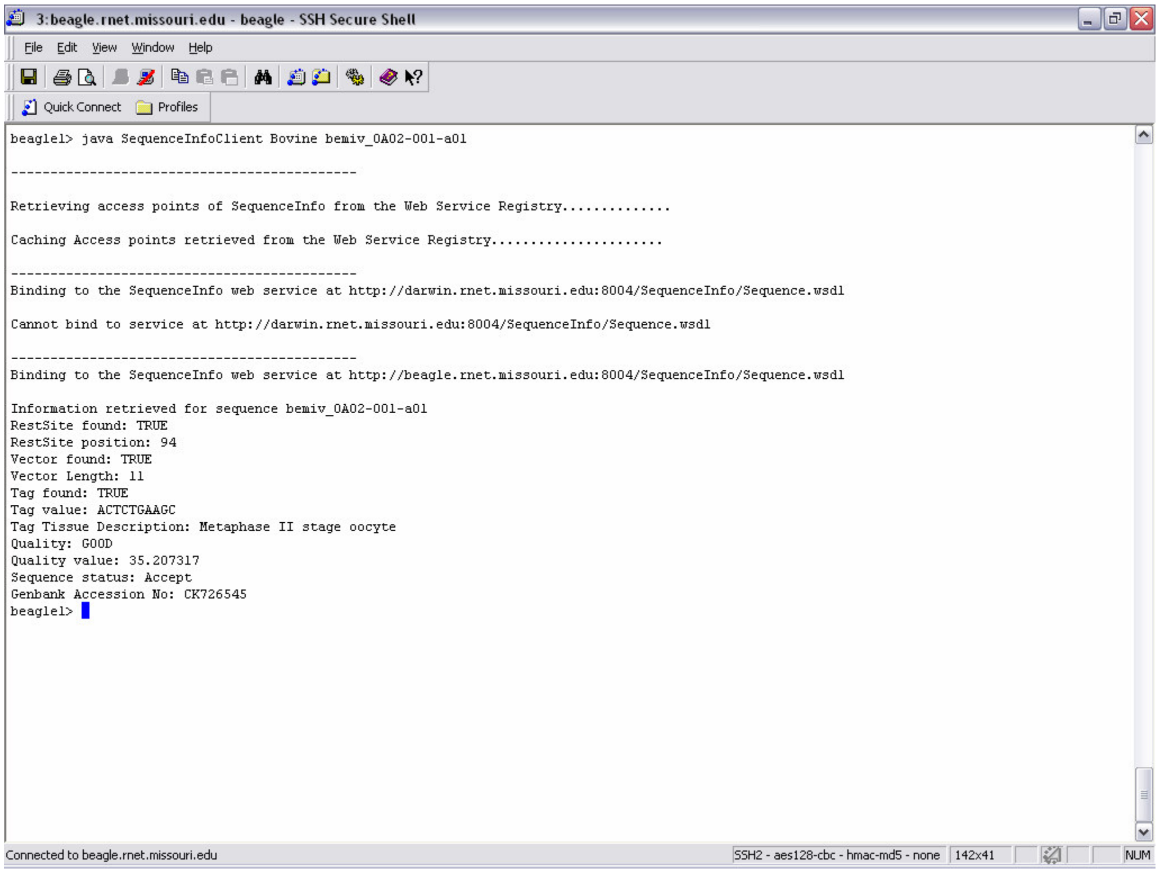


Figure 5.20: Client Accessing a Web Service Using the Web Service Registry

The client program can be written in such a way that if one access point of the web service ‘SequenceInfo’ is not available, it contacts the web service at another access point. If all the access points sent by the WsrInquireSoapServer are not available, there is a possibility that the web service ‘SequenceInfo’ is running on new servers (machines) with new access points. In this case, the client program calls the WSR again to get new access points of the web service ‘SequenceInfo’. Thus, as long as one instance of the web service ‘SequenceInfo’ is running and the WSR is updated with the access points, the client program finds a copy of the web service to communicate with.

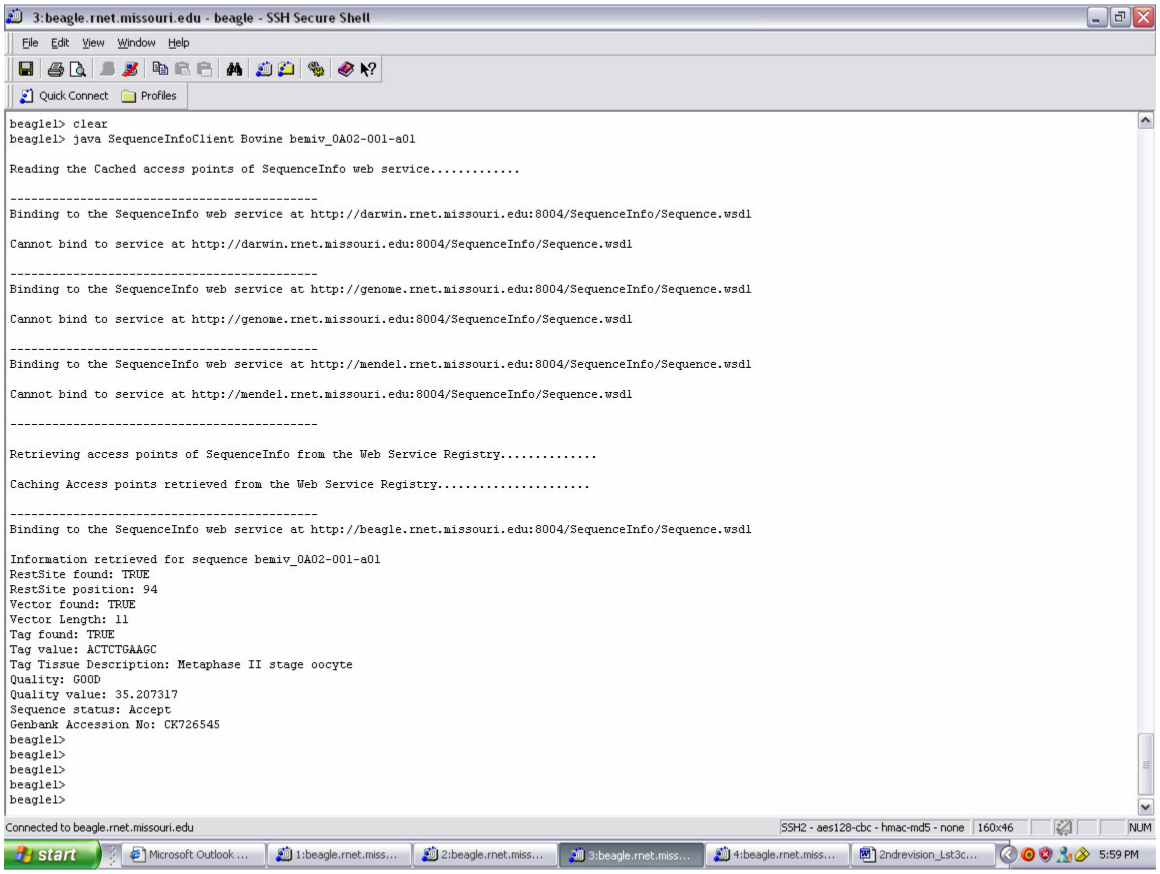
Figure 5.21 shows the first execution of the client program SequenceInfoClient to retrieve information about the quality of a sequence. The SequenceInfoClient calls the WsrInquireSoapServer to get the access points of the ‘SequenceInfo’ web service. The SequenceInfoClient caches the access points retrieved from the WSR. After it retrieves the access points, the SequenceInfoClient tries to call the ‘SequenceInfo’ web service at the first access point in the list received from the WSR. As shown in Figure 5.21, the SequenceInfoClient is not able to communicate with the web service at the first access point. With a failure to communicate with the first access point, the SequenceInfoClient tries to communicate with another instance of 'SequenceInfo'. This time the client program succeeds and it displays the information about the quality of the sequence. The project name and the sequence name are supplied at the command line to the SequenceInfoClient. Thus, the SequenceInfoClient was able to retrieve information about the quality of a sequence because there was at-least one instance of 'SequenceInfo' available to communicate with.


3:beagle.rnet.missouri.edu - beagle - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to beagle.rnet.missouri.edu
SSH2 - aes128-cbc - hmac-md5 - none
142x41
NUM

```
beagle1> java SequenceInfoClient Bovine bemiv_0A02-001-a01

--------------------------------------------

Retrieving access points of SequenceInfo from the Web Service Registry.............

Caching Access points retrieved from the Web Service Registry.....................

--------------------------------------------
Binding to the SequenceInfo web service at http://darwin.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

Cannot bind to service at http://darwin.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

--------------------------------------------
Binding to the SequenceInfo web service at http://beagle.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

Information retrieved for sequence bemiv_0A02-001-a01
RestSite found: TRUE
RestSite position: 94
Vector found: TRUE
Vector Length: 11
Tag found: TRUE
Tag value: ACTCTGAAGC
Tag Tissue Description: Metaphase II stage oocyte
Quality: GOOD
Quality value: 35.207317
Sequence status: Accept
Genbank Accession No: CK726545
beagle1>
```

Figure 5.21: First Execution of SequenceInfoClient to Access 'SequenceInfo' Web Service

Figure 5.22 shows the second execution of the SequenceInfoClient. The SequenceInfoClient checks to see if there are any cached access points to call 'SequenceInfo' web service. This time it reads the access points cached earlier instead of calling the WSR. As shown in the Figure 5.22, the SequenceInfoClient is not able to communicate with the web service at any of the cached access points and hence, it calls the WSR to retrieve new access points for ‘SequenceInfo’. The SequenceInfoClient updates the cached access points with the new access points received and communicates with the ‘SequenceInfo’ web service using one of the new access points.



```
beagle1> clear
beagle1> java SequenceInfoClient Bovine bemiv_0A02-001-a01

Reading the Cached access points of SequenceInfo web service............

--------------------------------------------
Binding to the SequenceInfo web service at http://darwin.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

Cannot bind to service at http://darwin.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

--------------------------------------------
Binding to the SequenceInfo web service at http://genome.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

Cannot bind to service at http://genome.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

--------------------------------------------
Binding to the SequenceInfo web service at http://mendel.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

Cannot bind to service at http://mendel.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

--------------------------------------------

Retrieving access points of SequenceInfo from the Web Service Registry.............

Caching Access points retrieved from the Web Service Registry.....................

--------------------------------------------
Binding to the SequenceInfo web service at http://beagle.rnet.missouri.edu:8004/SequenceInfo/Sequence.wsdl

Information retrieved for sequence bemiv_0A02-001-a01
RestSite found: TRUE
RestSite position: 94
Vector found: TRUE
Vector Length: 11
Tag found: TRUE
Tag value: ACTCTGAAGC
Tag Tissue Description: Metaphase II stage oocyte
Quality: GOOD
Quality value: 35.207317
Sequence status: Accept
Genbank Accession No: CK726545
beagle1>
beagle1>
beagle1>
beagle1>
beagle1>
```

Figure 5.22: Second Execution of SequenceInfoClient to Access ‘SequenceInfo’ Web Service

Thus the web service interface of the WSR, the multiple instances of a web service and a well written client program make the communication failures transparent to the user. The next chapter discusses the conclusions derived from this thesis and future work.

6. Conclusion

A recent development in Web technology called Web Services enables programs written using different programming languages, residing on heterogeneous systems to interact with each other. It uses Service Oriented Access Protocol (SOAP), a protocol used to represent data in a universally understood format, to send and interpret the data exchanged among the programs. SOAP can be sent over HTTP [4] (A transport protocol). The most commonly used transport protocol on the World Wide Web (WWW) is HTTP and thus, services (programs) can be built and deployed on the WWW to facilitate exchange of data on the WWW programmatically. These services deployed on the WWW are called web services. The information available on the WWW can be obtained from within a program written in any language and running on any platform.

Fault tolerance is crucial to services (programs) running on the WWW considering the vast number of users that might be using the services (programs). Fault-tolerance means the capability of a system or a service (program) to continue its operation in the event of failures. The main objective of this thesis was to devise a system to start and administer fault tolerant web services.

One of the important concerns, while designing the system, was to develop a system free from a single point of failure. If a system is built to administer fault-tolerant web services, it is required for the system also to be fault-tolerant. Thus, the design of the system to administer fault-tolerant web services comprises of components that do not have a single point of failure. This insures that the failure of a single component does not affect the operability of the system as a whole. This is achieved by having multiple instances of the same components of the system running on different servers (machines).

The system was also designed considering the advantages of distributed computing. Not only each component is replicated but also the tasks of the system are distributed over multiple machines, so that no single machine bears the load of running the entire system. Thus, the system also demonstrates distributed computing in addition to being fault-tolerant. To achieve fault-tolerance and implement distributed computing, the system is designed using both DCE and multicast.

The core component of the system (Web Service Manager) is implemented as a DCE server, such that there can be multiple instances of the DCE server running on different servers (machines) and any one server can be contacted to accomplish the task. The database accessed by the system to maintain and update information is replicated on different servers (machines) and is independent of the location of the other components of the system. Since there are multiple instances of the database, the components of the system communicate with the database using multicast. Multicast gives a host the ability to send out one packet to the network and have it received by a number of hosts (recipients) instead of sending out N different packets for N receivers. The components of the system have to send out just one message on the multicast address and the multicast communication ensures that the message is received by all the instances of the database. A multicast listener is installed on the database servers (machines) to listen and response to multicast messages originated from the other components of the system.

The Oracle [13] database is used by the fault-tolerant system to maintain and update information. An update to the database by the components of the system is performed by sending a multicast message that reaches all the database instances. A read operation is performed by remotely connecting to any one of the available database instances. To establish a remote connection, the components need to know the address of the database instance. The components send out a multicast message requesting address of a database instance to connect with. The multicast listeners running on different database servers (machines) respond with the address of the database instance. The components use the first response they receive to establish a remote connection with the database instance.

Multicast is used to communicate between the database instances. The interaction among the other components within the system, however, takes place using DCE-RPC, and HTTP and SOAP protocols. Thus, for the fault-tolerant system to be operable, only the database instances are required to be deployed on a multicast-enabled network. Multicast works on a Local Area Network (LAN), but for the multicast messages to reach the other segments of the network, the routers between the segments must be multicast-enabled.

The distributed nature of the system makes it possible to deploy the components on servers (machines) that best fit the need of the components. For example, the Web Service Manager can be run on a server (machine) that solely specializes in hosting DCE servers. Likewise, the database of the system can be deployed on servers (machines) that have strong and reliable back-up procedures to back-up the data.

Fault-tolerance is incorporated in web services (programs) by running multiple instances of the same web service on different servers (machines) such that at-least one instance is available at any given time to serve the requests from the users. This is accomplished with the help of the Web Service Manager, which is used to start and administer multiple instances of the same web service on different servers (machines). The Web Service Manager implements a set of remote procedures that are called by the administrator using a DCE client program (Wsmclient). The administrator is the person responsible of starting multiple instances of a web service. The DCE client program is a stand-alone program that locates the Web Service Manager and communicates with it. The Web Service Manager and the client program (Wsmclient) can be running on different servers (machines). As mentioned earlier, the Web Service Manager and the other components of the system communicate with the database of the system using multicast.

Thus, the DCE client program running on one machine communicates with the Web Service Manager on another machine using DCE and the Web Service Manager (or other components of the system) communicates with the database running on a different server (machine) using multicast. Thus, the task of the system as a whole is distributed on various servers (machines).

Once the multiple instances of a web service are started using the Web Service Manager, a periodic check is required to ensure the availability of the web service instances. Inherent to the task of the system is to start a new web service instance when a failure of a web service instance is detected. The periodic check and the replacement of a failed instance is done by a program called Monitor that monitors the instances periodically and starts a new instance to replace the failed instance.

Apart from designing and implementing the fault-tolerant system to start and administer fault-tolerant web services, the thesis also focused on building a Web Service Registry (WSR). A WSR is a repository of information about web services. It acts as a medium for web service providers to advertise their web services and thus, make web services available to the clients. Once the developers publish their web services, the clients can search the WSR for those web services using a search tool provided on the WSR. The search tool accepts keywords and other information from the clients and searches the database for the web services that match the criteria.

The interface of the WSR is made available in two forms. The first form is the conventional web interface form consisting of web pages, HTML forms and remote programs that are called using the CGI [5] mechanism. Collecting information from the user using HTML forms viewable in a web browser and calling remote programs on the WWW using CGI mechanism is the most conventional method to submit and retrieve information on the WWW. The required information is collected using HTML forms and sent to remote programs, which process the information sent and generate HTML web pages to be displayed back to the user in a web browser. The web interface of the WSR is deployed on two servers (machines) and can be started on more servers (machines). The address of the web interface of the WSR is stored in the database.

The second form is the form of web services that makes the operations of the WSR accessible programmatically. The web service interface consists of two web services that implement remote functions that can be called from a web service client. These web services are deployed as fault-tolerant web services by starting multiple instances on different servers (machines) using the fault-tolerant system designed. Thus, the thesis also demonstrates how an existing web interface can also be deployed as a fault-tolerant web service interface to make the same functionality available programmatically. The addresses of the web services that comprise the web service interface is made available on the web interface of the WSR.

Multicast is used to find the address(s) of the web interface form of the WSR by contacting the database instances. This is achieved by writing a client program in JAVA that sends out a multicast message to the database instances requesting the address(s) of the web interface. The client program has default address(s) of the web interface which it displays in failure to communicate with the database instances. The user can run the client program whenever he wants to gets the updated address(s) of the web interface of the WSR.

The WSR can be replicated and the users can access any instance of the WSR and register the web services. Regardless of the instance of the WSR where the user registers the web service, all the database instance(s) get updated using multicast. Thus, all the replicas of the WSR read the same dataset regardless of the database instance(s) they connect with. This allows the user to register the web service with the WSR easily accessible to the user and have it reflected in all the instances of the WSR.

The Web Service Registry built for this thesis project has a similar purpose to that of the Biomoby project [20]. The Biomoby project provides a platform to enable the exchange of Biological data among the various sources using web services. It is based on the concept of having a central registry where the web services are registered and the clients can search the registry for web services based on search criteria. Biomoby provides an Application Programming Interface (API) built in Java and Perl. The users can build Biomoby-compliant web services and register them with the Biomoby registry using the API. The users who want to register their web services with the Biomoby registery have to build the web services using the Java and Perl API provided.

For the WSR proposed in this thesis, the users are not required to adhere to any specific API to build the web services in order to register the web services with the WSR. The web services can be built using any API such as Microsoft's .NET or GLUE []. To register the web services, the users can visit the web interface of the WSR or use the web service interface of the WSR. The BioMoby project is specifically designed to support Biological data. Hence, the information required to be provided while registering a web service with the Biomoby registry is biologically oriented to help the search for a web service on the Biomoby registry. With the WSR proposed in this thesis any kind of web service can be registered as the information required to be provided to register a web service is not tailored for any specific web services, but the WSR can be customized to serve a specific community of users.

The web services built for this thesis are developed using GLUE [11], an Application Programming Interface (API) that provides the necessary interface to build web services in JAVA. The implementation of the fault-tolerant system required cross interaction between programs written in C and JAVA. This is accomplished using Java Native Interface (JNI) [12], an interface provided by the developers of JAVA to interact with a C program.

There are some areas that can improve the response time of the fault-tolerant system to start and administer fault-tolerant web services and make the system more secure. The DCE client program (Wsmclient) contacts the DCE Cell Directory Service (CDS) to get the hostnames of all the servers (machines) running the Web Service Manager as a DCE server. The hostnames are displayed to the administrator (user) running the client program and it is left to the discretion of the administrator to pick the server (machine) he wants to communicate with. A better approach would be to implement a load balancing algorithm in the DCE client program. The client program after retrieving the hostnames of the servers (machines) from CDS can contact each server (machine) to get information about the load running on each server. Based on the load information received from each server, the load balancing algorithm can determine a less busy server to communicate with. This way the task of the Web Service Manager is evenly distributed across all the servers (machines) running the Web Service Manager.

The data transferred between the HTML forms and the remote programs on the web interface of the WSR is unencrypted. This is not safe as data sent over the network can be intercepted by hackers and can be potentially harmful. This can be avoided by writing a script in the HTML form to encrypt the data before it is sent over the network. This approach has a drawback of having a custom encryption algorithm which can be viewed by looking at the source of the HTML page. A more secure solution is to use secure Web servers. This ensures that the data exchanged over the network is encrypted using standard and robust encryption algorithms. The disadvantage is that one has to get a certificate from a trusted authority. Information about setting up a secure web server can be found in Apache-SSL [15]. The search tool available on the WSR for searching web services can be made more exhaustive by including more options and criteria to search for web services.

This thesis has designed and implemented a fault-tolerant system to administer fault-tolerant web services. The system has been designed considering the importance and advantages of distributed computing. It has also demonstrated the use of DCE-RPC and multicast to provide interaction among the components of the system. The thesis also devised a WSR to provide access to the web services. The implementation of the fault-tolerant system and the WSR required learning different tools to build web services, to execute queries on Oracle database instances from within C programs and to interoperate programs written in C and JAVA. It required a considerable amount of time to learn to operate the different tools to successfully implement the fault-tolerant system and the Web Service Registry. This thesis also provides a guide on how to interoperate web services with legacy distributed applications built using DCE-RPC. It also exemplifies providing a web service interface to an existing legacy DCE application and bringing it into the greater arena of the WWW.

REFERENCES

1. Graham Glass, “Web Services Building Blocks for Distributed Systems”, Prentice-Hall, Inc., 2002.

2. Tom Savola, Mark Brown, John Jung, Bill Brandon, Robert Meegan, Kenneth Murphy, Jim O’Donnel, and Strephen R. Pietrowicz, “Special Edition Using HTML”, Que Corporation, 1996.

3. Ward Rosenberry, David Kenney, and Gerry Fisher, “Understanding DCE”, O’Reilly and Associates, Inc., 1992.

4. R.Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, January 1997.

5. Common Gateway Interface site, http://www.w3.org/CGI.

6. World Wide Web consortium site, http://www.w3.org.

7. Extensible Mark-up Language site, http://www.xml.org.

8. Uyless D. Black, “TCP/IP and related protocols”, McGraw-Hill, 1998.

9. David R. Kosiur, “IP multicasting : the complete guide to interactive corporate networks”, Wiley, 1998.

10. Kenn Scribner and Mark C.Stiver, “Applied SOAP: Implementing .NET XML Web Services”, Sams Publishing, 2002.

11. Webmethods site, http://www.webmethods.com.

12. Herbert Schildt, “Java 2: The Complete Reference”, fourth edition, McGraw-Hill, 2000.

13. Oracle site, http://www.oracle.com.

14. Unix Manual Pages site, http://www.rt.com/**man**.

15. Apache-SSL site, http://www.apache-ssl.org.

16. IBM site, http://www-306.ibm.com/software/solutions/webservices/uddi.

17. Microsoft, http://uddi.microsoft.com.

18. Raghu Ramakrishnan and Johannes Gehrke, “Database Management Systems”, second edition, McGraw-Hill, 1999.

19. Microsoft, “The Component Object Model Specification”, Draft 0.9, 1995.

20. BioMoby, http://www.biomoby.org.

Appendix A: Database Schema

It contains the Structured Query Language (SQL) commands to create database schema (tables) and execute some operations on the database schema created.

```
/* SQL commands to create the database schema (tables) on Oracle Database instances */

Create table WSM_Auth (
Username Varchar2(64),
Word     Varchar2(64),
Busid    Varchar2(64),
Primary Key (busid),
Unique (Username)
);

Create table WSR_Auth (
Username Varchar2(64),
Word     Varchar2(64),
BusId    Varchar2(64),
Address  Varchar2(256),
PhoneNo  Varchar2(32),
Email    Varchar2(128),
Primary Key (busid),
Unique (Username)
);

Create table WSR_Business(
BusID VARCHAR2(64),
ServiceID VARCHAR2(64),
PRIMARY KEY(serviceid),
Unique (BusID,serviceid),
Foreign key (busid) references WSR_Auth
ON DELETE CASCADE);

Create table WSM_Generic(
BusID VARCHAR2(64),
genericURL VARCHAR2(256),
Primary Key (BusId,genericURL),
Foreign key (busid) references WSM_Auth
ON DELETE CASCADE);

Create table WSM_Service(
BusID Varchar2(64),
Name  VARCHAR2(64) NOT NULL,
ServiceID  VARCHAR2(64) NOT NULL,
Primary key (serviceid),
Unique (serviceid,Name),
Foreign key (busid) references WSM_Auth
ON DELETE CASCADE);
```

```
Create table WSM_Replica(
serviceid VARCHAR2(64),
genericURL VARCHAR2(256),
serviceURL VARCHAR2(256),
timestamp  Date,
Primary Key (serviceId,genericURL),
Foreign key (serviceid) references WSM_Service
ON Delete Cascade);

Create table WSM_Standby(
serviceid VARCHAR2(64),
genericURL VARCHAR2(256),
Primary Key (serviceId,genericURL),
Foreign key (serviceid) references WSM_Service
ON Delete Cascade);

Create table WSM_Failure(
serviceid VARCHAR2(64),
serviceURL VARCHAR2(256),
timestamp date,
Primary Key (serviceId,serviceURL),
Foreign key (serviceid) references WSM_Service
ON Delete Cascade);

Create table WSR_Service(
serviceid VARCHAR2(64) NOT NULL,
Name  VARCHAR2(64) NOT NULL,
Descrp  VARCHAR2(256),
Keywords  VARCHAR2(256),
DocUrl  VARCHAR2(256),
Primary key (serviceid),
Foreign key (serviceid) references WSR_Business
ON Delete Cascade);


Create table WSR_Functions(
serviceid VARCHAR2(64) NOT NULL,
Name  VARCHAR2(128) NOT NULL,
InputParam  VARCHAR2(256),
OutputParam  VARCHAR2(256),
Foreign key (serviceid) references WSR_Service
ON Delete Cascade);
Create table WSR_AccessPoints(
serviceid VARCHAR2(64) NOT NULL,
Url  VARCHAR2(256) NOT NULL,
Foreign key (serviceid) references WSR_Service
ON Delete Cascade);

Create table Transactions(
Id VARCHAR2(64),
Status VARCHAR2(128),
Query  VARCHAR2(2048),
timestamp date,
Primary key (id)
)
partition by hash (Id)
(partition P1 tablespace USERS,
partition P2 tablespace USERS,
partition P3 tablespace USERS,
partition P4 tablespace USERS,
partition P5 tablespace USERS)
;

Create table DatabaseSync(
syncID  VARCHAR2(16),
syncflag VARCHAR2(16)
);

Create table Temp_transactions(
Id     VARCHAR2(64),
Query  VARCHAR2(2048),
timestamp date
);


/* SQL commands to create indexes on some of the data columns in the tables created */

create index inputParam_index on WSR_FUNCTIONS(INPUTPARAM)
indextype is ctxsys.context;

create index outParam_index on WSR_FUNCTIONS(OUTPUTPARAM)
indextype is ctxsys.context;

create index keywords_index on WSR_SERVICE(KEYWORDS)
indextype is ctxsys.context;

create index description_index on WSR_SERVICE(DESCRP)
indextype is ctxsys.context;

create index name_index on WSR_SERVICE(NAME)
indextype is ctxsys.context;


/* SQL commands to rebuild the indexes created */

alter index keywords_index rebuild;
alter index description_index rebuild;
alter name_index rebuild;

alter index inputparam_index rebuild;
alter index outputparam_index rebuild;
```

```
/* SQL commands to remove the database schema (tables) */

drop table wsr_accesspoints;
drop table wsr_functions;
drop table wsr_service;
drop table wsm_replica;
drop table wsm_standby;
drop table wsm_failure;
drop table wsm_service;
drop table wsr_business;
drop table wsr_auth;
drop table transactions;
drop table databaseSync;
drop table temp_transactions;
```

Appendix B: Source Code to Implement Fault-Tolerant System

It contains the source code of the components (programs) used to implement the fault-tolerant system.

Web Service Manger

1) Wsm.idl

```
/*************************************************************************
 * DCE Program Web Service Manager IDL file:  wsm.idl
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  Web Service Manager's IDL file
 *
 *
 * Change Log:
 *
 *************************************************************************/
[
   uuid(e24d317e-f683-11d6-8ddb-0004ac493e4b),
   version(1.0)
]
interface wsm
{
const long NameLen = 256;

typedef struct {
unsigned32 first;
unsigned32 size; /*
[ptr,max_is(size)]char *s2;*/
[first_is(first),length_is(size)] char array[0..1024];
}conarray;

typedef struct {
[string,ptr]        char *name;
[string,ptr]        char *word;
[string,ptr]        char *address;
[string,ptr]        char *phoneNo;
[string,ptr]        char *email;
}accountInfo;

typedef struct {
[string,ptr]     char *name;
[string,ptr]     char *description;
```

```
[string,ptr]      char *keywords;
[string,ptr]      char *DocumentUrl;
}serviceInfo;

/* Define Name Exported Function */
void serviceAction(
   [in]           handle_t  h,       /* DCE communication handle */
   [in,string,ptr] char *serviceName, /* NAme of the service */
   [in,ptr]                    serviceInfo *serviceDet,  /* details about the service */
   [in,string,ptr] char *paths, /*Paths of generic webservice*/
   [in,string,ptr] char *standbypaths, /*Paths of generic webservice*/
   [in,string]     char busid[64], /*Busid */
   [in]              unsigned32 *type,
   [in,out,ptr]        conarray *result,
   [out]       unsigned32 *status
           );

void getBusidInfo(
   [in]           handle_t  h,       /* DCE communication handle */
   [in,string,ptr]     char *Busid, /* Busid of the User*/
   [in,out,ptr]     accountInfo     *data,
   [in,out,ptr]           conarray *serviceName,
   [in,out,ptr]           conarray *serviceId,
   [out]             unsigned32 *status
);

void getServiceInfo(
   [in]           handle_t  h,       /* DCE communication handle */
   [in,string,ptr]     char *ServiceName, /* Name of the Service*/
   [in,string,ptr]     char *Busid, /* Business id*/
   [in,out,ptr]           conarray *replica,
   [in,out,ptr]        conarray *replicatime,
   [in,out,ptr]           conarray *standby,
   [out]              unsigned32 *status
);

void getAccounts(
   [in]           handle_t  h,       /* DCE communication handle */
   [in,out,ptr]           conarray *userName,
   [in,out,ptr]           conarray *userId,
   [in]              unsigned32 *type,
   [out]              unsigned32 *status
);


/* Define Name Exported Function */
void createAccount(
   [in]           handle_t  h,       /* DCE communication handle */
   [in,ptr]     accountInfo     *data,
   [in]              unsigned32 *type,
   [out,string]    char rstatus[256],
   [out]  unsigned32 *status
          );

void deleteAccount(
   [in]          handle_t  h,       /* DCE communication handle */
   [in,ptr]     accountInfo     *data,
   [in]               unsigned32 *type,
   [out]       unsigned32 *status
       );

void resetAccount(
   [in]          handle_t  h,       /* DCE communication handle */
   [in,ptr]     accountInfo     *data,
   [in]               unsigned32 *type,
   [out]       unsigned32 *status
       );


void updateAccount(
   [in]          handle_t  h,       /* DCE communication handle */
   [in,ptr]     accountInfo     *data,
   [in,string,ptr] char *busid, /*Busid */
   [out,string]        char rstatus[256]
       );

void updateServiceInfo(
   [in]          handle_t  h,       /* DCE communication handle */
   [in,string,ptr]     char *ServiceName, /* Name of the Service*/
   [in,ptr]     serviceInfo     *data,
   [in,string,ptr] char *busid, /*Busid */
   [out,string]        char rstatus[256]
       );

void updateServers(
   [in]       handle_t h,
   [in]               unsigned32 *type,
   [in,string,ptr]   char *serviceName,
   [in,string,ptr]   char *paths,
   [in,string,ptr] char *busid, /*Busid */
   [out,string]      char rstatus[256]
);

void deleteService(
   [in]          handle_t h,
   [in,string,ptr]   char *serviceName,
   [in,string,ptr]     char *busid, /*Busid */
   [out,string]      char rstatus[256]
);

void startStandbyServer(
   [in]          handle_t h,
   [in,string,ptr]   char *wsrid,
   [in,string,ptr]   char *standbypath,
   [out,string]      char rstatus[256]
);

void authen(
     [in]    handle_t h,
   [in,ptr]     accountInfo   *data,
   [out,string]      char rstatus[256],
   [out]       unsigned32 *status
);

void getGenericServers(
     [in]         handle_t h,
     [in,string,ptr]      char *busid,
     [in,out,ptr]    conarray *genpaths,
     [out]        unsigned32 *status
);

[idempotent]
void wsm_stop_server(
   [in]           handle_t   h,      /* DCE handle for communication  */
   [in,string,ptr] char       *uname, /* requesting user */
   [in,string,ptr] char       *valid, /* Validity flag */
   [out]          unsigned32      *rc     /* Return code */
   );

}
```

2) Wsmnms.h

```
/*************************************************************************
 * DCE Program Web Service Manager:  wsms.c - Server Main Program
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  Web Service Manager Server Program
 *
 *
 * Syntax: This server program is invoked with the command line:
 *           wsms [&]   (we should invoke program in background)
 *
 * Change Log:
 *
 *************************************************************************/
#include <jni.h>
/*
 * Standard C Include files
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <libgen.h>
#include <time.h>

#include <mcast.h>
/* Include DCE Definitions */
#include <dce/pthread.h>
#include <dce/rpc.h>  /* DCE RPC definitions */
#include <dce/uuid.h>  /* DCE uuid conversion definitions */

#include <wsm.h>   /* IDL generated Interface definition */
#include <wsmnms.h>   /* application names */

char            *pnm;       /* program name pointer for msgs */
uuid_t          objuuid;    /* binary version of object uuid string */
rpc_if_id_t     ifid;       /* interface identifier */
uuid_vector_t   *obj_uuid_vector;  /* object uuid vector for register */
rpc_binding_vector_t  *binding_vector;  /* Bindings for server */
unsigned32      estat;      /* server ending status code */
int         regstat;
int         debug;
char            entry_name[NMLEN];    /* Name Service Entry Name */
char            group_name[NMLEN];    /* Name Service Group Name */
char            annotation[NMLEN];    /* Server annotation in end point map */
char            hostname[NMLEN];      /* Name of our host */
char            s_objstr[40];         /* Server Object UUID String */
char                  db_String[256];

int sendMulticast(Message_t *msend, char *result);
void    cleanup();          /* unregister server routine */
void    thread_clean();          /* signal handler */
void      handler1();
char    *dcemsg();  /* DCE runtime error text message lookup */
char       *tmsg();

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

/**************************************************************************
 * DCEVECS -  Server Main Program
 **************************************************************************/
main(int argc, char **argv)
{
 unsigned32     st;        /* DCE status return codes on calls */
 int            i;         /* loop index */
 char           msg[144];  /* For time-stamped messages */
 Message_t msend;
 int            retcode;
 char                result[256];

memset(result,'\0',256);
memset(db_String,'\0',256);
/************************************************************************
 *   Start of main program
 ************************************************************************/
 pnm = strdup(basename(argv[0]));   /* get short name of program */

  sprintf(msg, "%s: Server starting execution...", pnm);
  tmsg(msg);

/* Trap certain non-DCE errors */
  signal(SIGHUP, thread_clean);
  signal(SIGINT, thread_clean);
  signal(SIGQUIT, thread_clean);
  signal(SIGTERM, thread_clean);
  signal(SIGALRM, handler1);
  regstat=0;
/******************** Register Server Interface ************************/

/* Get interface identifier for registration */
   rpc_if_inq_id(S_IFSPEC,  /* server interface specification */
              &ifid, /* server interface structure data */
              &st);  /* status code from inquire */
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: can't get server interface id\n\t%s",
            pnm, dcemsg(st));
      tmsg(msg);
      exit(1);
      }

/* Register the Server Interface with RPC Daemon */
   rpc_server_register_if(S_IFSPEC, &ifid.uuid, NULL, &st);
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: cannot register server interface\n\t%s",
            pnm, dcemsg(st));
      tmsg(msg);
      exit(1);
      }
   regstat = 1;
   sprintf(msg, "%s: server interface registered", pnm);
   tmsg(msg);  /* write time-stamped message */

/* Convert Object Type to its binary format */
   uuid_from_string((unsigned char *)OBJSTR, &objuuid, &st);
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: cannot convert object type\n\t%s", pnm, dcemsg(st));
      tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
      }

/* Register the object type with RPC runtime library */
   rpc_object_set_type(&objuuid, &ifid.uuid, &st);
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: can't set object type\n\t%s", pnm, dcemsg(st));
      tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
      }
      regstat = 2;

/****************** Create Server Binding Information ****************/
#ifdef ALLPROT
/*
 * Setup Runtime to use ALL protocol sequences
 */
   rpc_server_use_all_protseqs(rpc_c_protseq_max_calls_default, &st);
#else
/*
 * Setup Runtime to use just the datagram IP protocol
 */
   rpc_server_use_protseq((unsigned char *)"ncadg_ip_udp",
      rpc_c_protseq_max_calls_default, &st);
#endif

   if (st != rpc_s_ok) {
      sprintf(msg, "%s: can't setup server protocols", pnm);
      tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
      }
   sprintf(msg, "%s: server protocols registered", pnm);
   tmsg(msg);

   regstat=3;
/*
 * Get back what is actually supported as a protocol
 */
   rpc_server_inq_bindings(&binding_vector, &st);
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: can't inquire binding information\n\t%s",
            pnm, dcemsg(st));
     tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
     }

/*
 * Construct the object uuid vector to register with the Server
 */
   if (!(obj_uuid_vector=(uuid_vector_p_t)malloc(sizeof(uuid_vector_t)))) {
      sprintf(msg, "%s: can't get memory for object uuid", pnm);
      tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
      }
   obj_uuid_vector->count = 1;
   obj_uuid_vector->uuid[0] = &objuuid;

#ifdef NSREG
/********************Register with CDS Name services*********************/

if (strlen(entry_name) == 0) {
      gethostname(hostname,NMLEN);  /* our hostname */
      strcpy(entry_name, ENTNAME);  /* get NS entry name prefix */
      strcat(entry_name, hostname); /* create NS entry name */
      }
   if (strlen(group_name) == 0) {
      strcpy(group_name, GRPNAME);  /* create NS group name */
      }
   if (strlen(annotation) == 0) {
      strcpy(annotation, EPNAME);   /* annotation name */
      }

/*
 * Export the server bindings to the directory service
 */

   rpc_ns_binding_export(rpc_c_ns_syntax_default,
                     (unsigned_char_t *) entry_name,
                     S_IFSPEC,
                     binding_vector,
                     obj_uuid_vector,
                     &st);
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: can't export server binding to NS", pnm);
      tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
      }
          regstat=4;

      sprintf(msg, "%s: server bindings exported to NS for\n\t%s",
          pnm, (char *) entry_name);
      tmsg(msg);

/*
 * Add Entry to server's Server Group in Name Service
 */
   rpc_ns_group_mbr_add(rpc_c_ns_syntax_default,
                     (unsigned_char_t *) group_name,
                     rpc_c_ns_syntax_default,
                     (unsigned_char_t *) entry_name,
                     &st);
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: can't add member to NS group", pnm);
      tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
      }
          regstat=5;
   sprintf(msg, "%s: server added to NS group\n\t%s",
          pnm, (char *) group_name);
   tmsg(msg);

/*********************************************************************/
#endif

/*
 * Register the dynamic endpoints that were created
 */
   rpc_ep_register(S_IFSPEC,  /* server if_spec */
                 binding_vector,   /* binding vector */
                 obj_uuid_vector,  /* object uuid */
                 (unsigned_char_t *) annot,  /* annotation string */
                 &st);  /* status returned from register */
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: can't register server endpoints\n\t%s",
            pnm, dcemsg(st));
      tmsg(msg);
      cleanup(st);  /* write error message and unregister everything */
          }
          regstat=6;
   sprintf(msg, "%s: server endpoints registered", pnm);
   tmsg(msg);

/*
 * walk the binding vector and show the protocols registered
 */
 for (i = 0; i < binding_vector->count; i++) {
      st= p_handle(binding_vector->binding_h[i]);
      if (st != rpc_s_ok) {
          sprintf(msg, "%s: can't print the server bindings\n\t%s",
                pnm, dcemsg(st));
          tmsg(msg);
          cleanup(st);  /* write error message and unregister everything */
          }
      }  /* end for to walk the binding vector */

/**********************************************************************
 * Server will sit here and wait for client requests until an external
 * signal (e.g., SIGTERM) causes it to shutdown.
 *
 * The server can handle up to system maximum simultaneous requests.
 ***********************************************************************/
   sprintf(msg, "%s: server starting to listen...", pnm);
   tmsg(msg);

/* send Multicast message to get the db_String */
           memset ((void *)&msend, 0, sizeof(Message_t));

        memset(msend.msg,'\0',MaxMsg);
        msend.type=ntohl(DBASE_QUERY);

           retcode = sendMulticast(&msend,result);
if ( retcode ==0)
           {
           strcpy(db_String,result);
                     sprintf(msg,"Found a database to connect with having DbString
%s\n",db_String);
             tmsg(msg);
           }
else
           {
                     sprintf(msg,"Can't find a database to connect with\n");
           tmsg(msg);
                     cleanup(st);  /* write error message and unregister everything */
           }

TRY
   rpc_server_listen(rpc_c_listen_max_calls_default,
                &st);   /* Listen for client requests */
   if (st != rpc_s_ok) {
     sprintf(msg, "%s: abnormal listen failure", pnm); /* tell of failure */
     tmsg(msg);
     thread_clean(st);  /* cleanup the threads before unregister */
     }
CATCH_ALL
/* Any errors that occur will have server unregister its endpoints */
   sprintf(msg,"%s: Server caught an error...\n\t%s", pnm, dcemsg(st));
   tmsg(msg);
   thread_clean(st);  /* cleanup threads before unregister */
ENDTRY
   cleanup(estat);  /* call cleanup to unregister server */
   exit(0);   /* will never get here */
}  /* end main() */


/***********************************************************************
 * Cleanup routine to unregister the server prior to termination
 *
 * Parameters:
 *   stin  - DCE or signal status code that brought us here
 ***********************************************************************/
void cleanup(int stin)
```

```
{
  unsigned32  st;         /* status code for DCE calls */
  char        msg[144];   /* For time-stamped messages */

/*
 * Tell them how we got here
 */
  if (stin < 31)  /* was it a Unix signal? */
    sprintf(msg, "%s: server cleanup - Unix signal=%d", pnm, stin);
  else
    sprintf(msg, "%s: server cleanup - Status=%08x\n\t%s",
         pnm, stin, dcemsg(stin));

/***************************************************************************
 * Unregister the server and terminate
 ***************************************************************************/
switch (regstat) {
  case 6:
    /*
     * UN-Register the dynamic endpoints that were created
     */
      rpc_ep_unregister(S_IFSPEC, binding_vector,
         obj_uuid_vector, &st);
      if (st != rpc_s_ok) {
        sprintf(msg,
             "%s(cleanup): can't unregister server endpoints\n\t%s",
             pnm, dcemsg(st));
        tmsg(msg); /* write message */
        }
      else {
        sprintf(msg, "%s(cleanup): server endpoints unregistered", pnm);
        tmsg(msg); /* write message */
        }

#ifdef NSREG
  case 5:
    /*
     * Remove member from its Server Group in Name Service
     */
      rpc_ns_group_mbr_remove(rpc_c_ns_syntax_dce,
                    (unsigned_char_t *) group_name,
                    rpc_c_ns_syntax_dce,
                    (unsigned_char_t *) entry_name,
                    &st);
      if (st != rpc_s_ok) {
        sprintf(msg,
             "%s(cleanup): can't remove %s from\n\t%s NS group\n\t%s",
             pnm, (char *) entry_name, (char *) group_name,
             dcemsg(st));
        tmsg(msg);
        }
        else {
          sprintf(msg, "%s(cleanup): removed %s from\n\t%s NS group",
                pnm, (char *) entry_name, (char *) group_name);
          tmsg(msg);
          }
  case 4:
     /*
      * UN-Export the server bindings from the directory service
      */
        rpc_ns_binding_unexport(rpc_c_ns_syntax_dce,
                        (unsigned_char_t *) entry_name,
                        S_IFSPEC,
                        obj_uuid_vector,
                        &st);
        if (st != rpc_s_ok) {
          sprintf(msg,
                "%s(cleanup): can't unexport server binding from NS\n\t%s",
                pnm, dcemsg(st));
          tmsg(msg);
          }
        else {
          sprintf(msg, "%s(cleanup): server bindings unexported from NS",
            pnm);
          tmsg(msg);
          }
#endif
  case 3:
  case 2:
        rpc_object_set_type(&objuuid, NULL, &st);
        if (st != rpc_s_ok) {
          sprintf(msg, "%s(cleanup): can't unregister object type", pnm);
          tmsg(msg);
          }
        else {
          sprintf(msg, "%s(cleanup): server object type unregistered", pnm);
          tmsg(msg);
          }

  case 1:
        rpc_server_unregister_if(S_IFSPEC, NULL, &st);
        if (st != rpc_s_ok) {
          sprintf(msg,
                "%s(cleanup): can't unregister server interface\n\t%s",
                pnm, dcemsg(st));
          tmsg(msg);
          }
        else {
          sprintf(msg, "%s(cleanup): server interface unregistered", pnm);
          tmsg(msg);
          }
  default:
        sprintf(msg, "%s(cleanup): registration status=%d", pnm, regstat);
        tmsg(msg);
        break;
      }  /* end switch */

  sprintf(msg, "%s(cleanup): server has unregistered", pnm);
  tmsg(msg);



  sprintf(msg, "%s: Server shutting down...", pnm);
  tmsg(msg);
  exit(stin);        /* and go home to bed */
}  /* end cleanup() */


/************************************************************************
 * Thread_clean - cleanup threads by telling server to stop listening
 *
 * This code gets activated by trapping a non-DCE signal from the server
 * environment.  For example, a SIGTERM signal is issued to the process.
 ************************************************************************/
void thread_clean(unsigned32 stin)
{
  int         i;
  unsigned32  st;
  char        msg[144];   /* For time-stamped messages */

/* output a message to say why we got here */
  sprintf(msg, "th_clean: error code = %d", stin);

/* Tell the server to stop listening so we can shut down */
  rpc_mgmt_stop_server_listening(NULL, &st);
  if (st != rpc_s_ok) {
    sprintf(msg, "th_clean: can't stop server listening: %d\n\t%s",
          i, dcemsg(st));
    }
  else tmsg("th_clean: stopped server listening");
  sprintf(msg, "th_clean: exiting cleanup of threads\n");

  estat = stin;  /* save ending status for cleanup() usage */
  cleanup(stin);
}  /* end thread_clean() */


void handler1(int x)
{
        printf("ALarm receive\n");
}
```

3) Wsms.c

```
/*********************************************************************
 * DCE Program Web Service Manager:  wsms.c - Server Main Program
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  Web Service Manager Server Program
 *
 *
 * Syntax: This server program is invoked with the command line:
 *          wsms [&]   (we should invoke program in background)
 *
 * Change Log:
 *
 *********************************************************************/
#include <jni.h>
/*
 * Standard C Include files
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <libgen.h>
#include <time.h>

#include <mcast.h>
/* Include DCE Definitions */
#include <dce/pthread.h>
#include <dce/rpc.h>  /* DCE RPC definitions */
#include <dce/uuid.h>  /* DCE uuid conversion definitions */

#include <wsm.h>   /* IDL generated Interface definition */
#include <wsmnms.h>   /* application names */

char            *pnm;       /* program name pointer for msgs */
uuid_t          objuuid;    /* binary version of object uuid string */
rpc_if_id_t     ifid;       /* interface identifier */
uuid_vector_t   *obj_uuid_vector;  /* object uuid vector for register */
rpc_binding_vector_t  *binding_vector;  /* Bindings for server */
unsigned32      estat;      /* server ending status code */
int         regstat;
int         debug;
char            entry_name[NMLEN];    /* Name Service Entry Name */
char            group_name[NMLEN];    /* Name Service Group Name */
char            annotation[NMLEN];    /* Server annotation in end point map */
char           hostname[NMLEN];      /* Name of our host */
char           s_objstr[40];         /* Server Object UUID String */
char                db_String[256];

int sendMulticast(Message_t *msend, char *result);
void    cleanup();          /* unregister server routine */
void    thread_clean();         /* signal handler */
void      handler1();
char    *dcemsg();  /* DCE runtime error text message lookup */
char       *tmsg();

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

/*************************************************************************
 * DCEVECS -  Server Main Program
 *************************************************************************/
main(int argc, char **argv)
{
  unsigned32     st;         /* DCE status return codes on calls */
  int           i;          /* loop index */
  char           msg[144];   /* For time-stamped messages */
  Message_t msend;
  int          retcode;
  char                result[256];

memset(result,'\0',256);
memset(db_String,'\0',256);
/**********************************************************************
 *   Start of main program
 **********************************************************************/
 pnm = strdup(basename(argv[0]));   /* get short name of program */

  sprintf(msg, "%s: Server starting execution...", pnm);
  tmsg(msg);

/* Trap certain non-DCE errors */
  signal(SIGHUP, thread_clean);
  signal(SIGINT, thread_clean);
  signal(SIGQUIT, thread_clean);
  signal(SIGTERM, thread_clean);
  signal(SIGALRM, handler1);
  regstat=0;
/******************* Register Server Interface ************************/

/* Get interface identifier for registration */
   rpc_if_inq_id(S_IFSPEC,  /* server interface specification */
              &ifid, /* server interface structure data */
```

```
            &st);  /* status code from inquire */
  if (st != rpc_s_ok) {
    sprintf(msg, "%s: can't get server interface id\n\t%s",
          pnm, dcemsg(st));
    tmsg(msg);
    exit(1);
    }

/* Register the Server Interface with RPC Daemon */
  rpc_server_register_if(S_IFSPEC, &ifid.uuid, NULL, &st);
  if (st != rpc_s_ok) {
    sprintf(msg, "%s: cannot register server interface\n\t%s",
          pnm, dcemsg(st));
    tmsg(msg);
    exit(1);
    }
  regstat = 1;
  sprintf(msg, "%s: server interface registered", pnm);
  tmsg(msg);  /* write time-stamped message */

/* Convert Object Type to its binary format */
  uuid_from_string((unsigned char *)OBJSTR, &objuuid, &st);
  if (st != rpc_s_ok) {
    sprintf(msg, "%s: cannot convert object type\n\t%s", pnm, dcemsg(st));
    tmsg(msg);
    cleanup(st);  /* write error message and unregister everything */
    }

/* Register the object type with RPC runtime library */
  rpc_object_set_type(&objuuid, &ifid.uuid, &st);
  if (st != rpc_s_ok) {
    sprintf(msg, "%s: can't set object type\n\t%s", pnm, dcemsg(st));
    tmsg(msg);
    cleanup(st);  /* write error message and unregister everything */
    }
    regstat = 2;

/****************** Create Server Binding Information ****************/
#ifdef ALLPROT
/*
 * Setup Runtime to use ALL protocol sequences
 */
  rpc_server_use_all_protseqs(rpc_c_protseq_max_calls_default, &st);
#else
/*
 * Setup Runtime to use just the datagram IP protocol
 */
  rpc_server_use_protseq((unsigned char *)"ncadg_ip_udp",
      rpc_c_protseq_max_calls_default, &st);
#endif
```

```
  if (st != rpc_s_ok) {
     sprintf(msg, "%s: can't setup server protocols", pnm);
     tmsg(msg);
     cleanup(st);  /* write error message and unregister everything */
     }
  sprintf(msg, "%s: server protocols registered", pnm);
  tmsg(msg);

  regstat=3;
/*
 * Get back what is actually supported as a protocol
 */
  rpc_server_inq_bindings(&binding_vector, &st);
  if (st != rpc_s_ok) {
     sprintf(msg, "%s: can't inquire binding information\n\t%s",
             pnm, dcemsg(st));
    tmsg(msg);
     cleanup(st);  /* write error message and unregister everything */
     }

/*
 * Construct the object uuid vector to register with the Server
 */
  if (!(obj_uuid_vector=(uuid_vector_p_t)malloc(sizeof(uuid_vector_t)))) {
     sprintf(msg, "%s: can't get memory for object uuid", pnm);
     tmsg(msg);
     cleanup(st);  /* write error message and unregister everything */
     }
  obj_uuid_vector->count = 1;
  obj_uuid_vector->uuid[0] = &objuuid;

#ifdef NSREG
/********************Register with CDS Name services*********************/

if (strlen(entry_name) == 0) {
     gethostname(hostname,NMLEN);  /* our hostname */
     strcpy(entry_name, ENTNAME);  /* get NS entry name prefix */
     strcat(entry_name, hostname); /* create NS entry name */
     }
  if (strlen(group_name) == 0) {
     strcpy(group_name, GRPNAME);  /* create NS group name */
     }
  if (strlen(annotation) == 0) {
     strcpy(annotation, EPNAME);   /* annotation name */
     }

/*
 * Export the server bindings to the directory service
 */

  rpc_ns_binding_export(rpc_c_ns_syntax_default,
                   (unsigned_char_t *) entry_name,
                   S_IFSPEC,
                   binding_vector,
                   obj_uuid_vector,
                   &st);
  if (st != rpc_s_ok) {
     sprintf(msg, "%s: can't export server binding to NS", pnm);
     tmsg(msg);
     cleanup(st);  /* write error message and unregister everything */
     }
         regstat=4;

     sprintf(msg, "%s: server bindings exported to NS for\n\t%s",
         pnm, (char *) entry_name);
     tmsg(msg);

/*
 * Add Entry to server's Server Group in Name Service
 */
  rpc_ns_group_mbr_add(rpc_c_ns_syntax_default,
                   (unsigned_char_t *) group_name,
                   rpc_c_ns_syntax_default,
                   (unsigned_char_t *) entry_name,
                   &st);
  if (st != rpc_s_ok) {
     sprintf(msg, "%s: can't add member to NS group", pnm);
     tmsg(msg);
     cleanup(st);  /* write error message and unregister everything */
     }
         regstat=5;
  sprintf(msg, "%s: server added to NS group\n\t%s",
         pnm, (char *) group_name);
  tmsg(msg);

/***********************************************************************/
#endif

/*
 * Register the dynamic endpoints that were created
 */
  rpc_ep_register(S_IFSPEC,  /* server if_spec */
              binding_vector,   /* binding vector */
              obj_uuid_vector,  /* object uuid */
              (unsigned_char_t *) annot,  /* annotation string */
              &st);  /* status returned from register */
  if (st != rpc_s_ok) {
     sprintf(msg, "%s: can't register server endpoints\n\t%s",
             pnm, dcemsg(st));
     tmsg(msg);
     cleanup(st);  /* write error message and unregister everything */
         }
```

```
            regstat=6;
   sprintf(msg, "%s: server endpoints registered", pnm);
   tmsg(msg);

/*
 * walk the binding vector and show the protocols registered
 */
  for (i = 0; i < binding_vector->count; i++) {
      st= p_handle(binding_vector->binding_h[i]);
      if (st != rpc_s_ok) {
         sprintf(msg, "%s: can't print the server bindings\n\t%s",
                 pnm, dcemsg(st));
         tmsg(msg);
         cleanup(st);  /* write error message and unregister everything */
         }
      }  /* end for to walk the binding vector */

/************************************************************************
 * Server will sit here and wait for client requests until an external
 * signal (e.g., SIGTERM) causes it to shutdown.
 *
 * The server can handle up to system maximum simultaneous requests.
 ************************************************************************/
   sprintf(msg, "%s: server starting to listen...", pnm);
   tmsg(msg);

/* send Multicast message to get the db_String */
            memset ((void *)&msend, 0, sizeof(Message_t));

        memset(msend.msg,'\0',MaxMsg);
        msend.type=ntohl(DBASE_QUERY);

            retcode = sendMulticast(&msend,result);
if ( retcode ==0)
            {
            strcpy(db_String,result);
                        sprintf(msg,"Found a database to connect with having DbString
%s\n",db_String);
              tmsg(msg);
            }
else
            {
                        sprintf(msg,"Can't find a database to connect with\n");
            tmsg(msg);
                        cleanup(st);  /* write error message and unregister everything */
            }

TRY
   rpc_server_listen(rpc_c_listen_max_calls_default,
                 &st);   /* Listen for client requests */
   if (st != rpc_s_ok) {
      sprintf(msg, "%s: abnormal listen failure", pnm); /* tell of failure */
      tmsg(msg);
      thread_clean(st);  /* cleanup the threads before unregister */
      }
CATCH_ALL
/* Any errors that occur will have server unregister its endpoints */
   sprintf(msg,"%s: Server caught an error...\n\t%s", pnm, dcemsg(st));
   tmsg(msg);
   thread_clean(st);  /* cleanup threads before unregister */
ENDTRY
   cleanup(estat);  /* call cleanup to unregister server */
   exit(0);   /* will never get here */
}  /* end main() */


/************************************************************************
 * Cleanup routine to unregister the server prior to termination
 *
 * Parameters:
 *   stin  - DCE or signal status code that brought us here
 ************************************************************************/
void cleanup(int stin)
{
   unsigned32  st;         /* status code for DCE calls */
   char        msg[144];   /* For time-stamped messages */

/*
 * Tell them how we got here
 */
   if (stin < 31)  /* was it a Unix signal? */
      sprintf(msg, "%s: server cleanup - Unix signal=%d", pnm, stin);
   else
      sprintf(msg, "%s: server cleanup - Status=%08x\n\t%s",
            pnm, stin, dcemsg(stin));

/************************************************************************
 * Unregister the server and terminate
 ************************************************************************/
switch (regstat) {
   case 6:
      /*
       * UN-Register the dynamic endpoints that were created
       */
        rpc_ep_unregister(S_IFSPEC, binding_vector,
            obj_uuid_vector, &st);
        if (st != rpc_s_ok) {
          sprintf(msg,
                "%s(cleanup): can't unregister server endpoints\n\t%s",
                pnm, dcemsg(st));
          tmsg(msg); /* write message */
          }
        else {
          sprintf(msg, "%s(cleanup): server endpoints unregistered", pnm);
          tmsg(msg); /* write message */
          }

#ifdef NSREG
  case 5:
    /*
     * Remove member from its Server Group in Name Service
     */
        rpc_ns_group_mbr_remove(rpc_c_ns_syntax_dce,
                        (unsigned_char_t *) group_name,
                        rpc_c_ns_syntax_dce,
                        (unsigned_char_t *) entry_name,
                        &st);
        if (st != rpc_s_ok) {
          sprintf(msg,
               "%s(cleanup): can't remove %s from\n\t%s NS group\n\t%s",
               pnm, (char *) entry_name, (char *) group_name,
               dcemsg(st));
          tmsg(msg);
          }
        else {
          sprintf(msg, "%s(cleanup): removed %s from\n\t%s NS group",
               pnm, (char *) entry_name, (char *) group_name);
          tmsg(msg);
          }
  case 4:
    /*
     * UN-Export the server bindings from the directory service
     */
        rpc_ns_binding_unexport(rpc_c_ns_syntax_dce,
                        (unsigned_char_t *) entry_name,
                        S_IFSPEC,
                        obj_uuid_vector,
                        &st);
        if (st != rpc_s_ok) {
          sprintf(msg,
               "%s(cleanup): can't unexport server binding from NS\n\t%s",
               pnm, dcemsg(st));
          tmsg(msg);
          }
        else {
          sprintf(msg, "%s(cleanup): server bindings unexported from NS",
            pnm);
          tmsg(msg);
          }
#endif
  case 3:
  case 2:
        rpc_object_set_type(&objuuid, NULL, &st);
        if (st != rpc_s_ok) {
          sprintf(msg, "%s(cleanup): can't unregister object type", pnm);
          tmsg(msg);
          }
        else {
          sprintf(msg, "%s(cleanup): server object type unregistered", pnm);
          tmsg(msg);
          }

  case 1:
        rpc_server_unregister_if(S_IFSPEC, NULL, &st);
        if (st != rpc_s_ok) {
          sprintf(msg,
                "%s(cleanup): can't unregister server interface\n\t%s",
                pnm, dcemsg(st));
          tmsg(msg);
          }
        else {
          sprintf(msg, "%s(cleanup): server interface unregistered", pnm);
          tmsg(msg);
          }
  default:
        sprintf(msg, "%s(cleanup): registration status=%d", pnm, regstat);
        tmsg(msg);
        break;
      }  /* end switch */

  sprintf(msg, "%s(cleanup): server has unregistered", pnm);
  tmsg(msg);



  sprintf(msg, "%s: Server shutting down...", pnm);
  tmsg(msg);
  exit(stin);        /* and go home to bed */
}  /* end cleanup() */

/************************************************************************
 * Thread_clean - cleanup threads by telling server to stop listening
 *
 * This code gets activated by trapping a non-DCE signal from the server
 * environment.  For example, a SIGTERM signal is issued to the process.
 ************************************************************************/
void thread_clean(unsigned32 stin)
{
  int         i;
  unsigned32  st;
  char        msg[144];   /* For time-stamped messages */

/* output a message to say why we got here */
  sprintf(msg, "th_clean: error code = %d", stin);
```

```
/* Tell the server to stop listening so we can shut down */
  rpc_mgmt_stop_server_listening(NULL, &st);
  if (st != rpc_s_ok) {
    sprintf(msg, "th_clean: can't stop server listening: %d\n\t%s",
         i, dcemsg(st));
    }
  else tmsg("th_clean: stopped server listening");
  sprintf(msg, "th_clean: exiting cleanup of threads\n");

  estat = stin;  /* save ending status for cleanup() usage */
  cleanup(stin);
}  /* end thread_clean() */


void handler1(int x)
{
        printf("ALarm receive\n");
}
```

4) Wsm_mgr.c

```
#pragma options showinc source
/**************************************************************************
 * DCE Program wsm_mgr:  wsm_mgr.c - Functions in the DCEVEC Interface
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  Implement the DCE interface for the web service manager (WSM).
 *
 * Function: Define the interface functions in the WSM Server.
 *          This code is invoked directly by the Server (WSMS.C)
 *
 **************************************************************************/
 #pragma  options nosource
#include <stdio.h>
#include "wsm.h"  /* IDL generated Interface Definition */
#include <dirent.h>
 #include <time.h>
#include<sys/stat.h>
#include <string.h>
#include <sys/file.h>
#include <errno.h>
#include "mcast.h"
#include "database.h"
#include <signal.h>
#include <unistd.h>
```

```
#pragma  options source
#include <wsmnms.h> /* application names */
#define  MAX_CHAR 2048
#define MAX_WSR  128
#define  logfile "servlog"

extern int debug;
char *method[5] = {"","start","start","stop","status"};
char*       tmsg();     /* produce time-stamped messages on stdout */

int searchfile(char * ,idl_char *);        /* To search a file in a given directory */
void          handler();
int callGenericService(char *method,char *WSRid, char *pathGeneric,char *result);
int createService(idl_char *serviceName,serviceInfo *serviceDet,char *busid,char *WSRid, char
**pathGeneric,char **standbypaths,int noGenPath, int noStandbyPath, char **result,int type);
int stopStatusService(idl_char *serviceName,char *busid, char *WSRid, char **pathGeneric,int
noGenPath, char **result, int type);
int sendMulticast(Message_t *msend, char *result);
int fetch(char *query,char **data[],int noColumns);
int standbyStart(char *wsrid,char *standbypath,char *result);
int authenticateAdmin(char *Username,char *Word,char *rstatus);
void getuuid(char *,char *);



/**********************************************************************
 * Remote create function in the DCEVEC Server Interface
 *
 * Purpose:  To create an entry in the database for the new account.
 * Function: To form a SQL query to insert a row in the respective table in the database based on
the information
 *          sent by the client. Send the SQL query on a multicast address where the multicast
listeners will execute
 *             the query on the respective local servers and send the status of execution of the query
back to the caller.
 * Returns:  return code indicating the status of execution.
 *          return 0 if the query is successfully executed to insert a row and also send an unique-id
back to the client.
 *          return 1 if error and  send the error String back to the client.
 **********************************************************************/
void createAccount(
      handle_t h,
      accountInfo *data, /* data send by the client */
          unsigned32 *type,       /* type of account, i.e. WSM or WSR */
      idl_char  *result , /* Status string sent to the client */
          unsigned32  *status /* return code sent to the client */
)

{
          int retcode;    /* Return code from the sendMulticast function */
          char resultsend[256]; /* Return string from the sendMulticast function */
```

```
        Message_t msend;    /* Message structure sent on the multicast address */
        char busid[64];   /* unique busid for each account */
        char msg[256];
        char SQLstat[512];
        char *temp;
        char transactionid[64];
    time_t calltime;

    time(&calltime);


        /*sprintf(msg,"createAccount called for user %s\n",data->name);
    tmsg(msg);*/

        memset(resultsend,'\0',256);
        memset(busid,'\0',64);
        memset(transactionid,'\0',64);

        getuuid(busid,data->name);

        sprintf(transactionid,"%d%s",calltime,busid);

        memset ((void *)&msend, 0, sizeof(Message_t));

    memset(msend.msg,'\0',MaxMsg);

    msend.type=ntohl(UPDATE); /* Set the type of the multicast message */
    msend.App=htonl(WSM);  /* Which application */

    sprintf(msend.msg,"%s,",transactionid);
        sprintf(SQLstat,"BEGIN ");
    strcat(msend.msg,SQLstat);


        /* Prepare the SQL query */
        sprintf(SQLstat,"INSERT INTO %s (%s,%s,%s,%s,%s,%s) VALUES
('%s','%s','%s','%s','%s','%s');",WSR_AUTH,WSR_AUTH_NAME,WSR_AUTH_WORD,WSR_
AUTH_BUSID,WSR_AUTH_ADD,WSR_AUTH_PHONE,WSR_AUTH_EMAIL, data-
>name,data->word,busid,data->address,data->phoneNo,data->email);

        strcat(msend.msg,SQLstat);

        if(*type==WSM)
        {
                sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s','%s');",WSM_AUTH,
data->name,data->word,busid);
                strcat(msend.msg,SQLstat);
        }

        sprintf(SQLstat,"END;");
        strcat(msend.msg,SQLstat);
        if(debug)printf("In CreateAccount. query *%s*\n",msend.msg);

        /* call the function to multicast the message to the listeners */
        if(sendMulticast(&msend,resultsend))
        {
                strcpy(result,"Error in creating account. Contact administrator");
                *status=1;    /* set the status code to indicate error */
        }

        else
        {
                if(!strcmp(resultsend,"OK"))
                {
                        *status=0;    /*set the status code to indicate successful creation of the
account */
                        strcpy(result,busid);    /* Send the unique-id to the client */
                }
                else
                {
                        *status=1;
                        if(strstr(resultsend,"ORA-00001"))  /* if an account with the given
username already exists */
                        strcpy(result,"ERROR:Account already exists with current username");
                        else
                        strcpy(result,"ERROR:Error in creating account. Contact administrator");
                        printf("HEREQ\n");
                }
        }

        if(debug) printf("Exiting createAccount with status %d and result string
%s\n",*status,result);

        /*sprintf(msg,"createAccount for user %s exited with status %d and result string
%s\n",data->name,*status,result);
     tmsg(msg);*/

        return;
}

/*******************************************************************
 * Remote delete account function in the DCEVEC Server Interface
 *
 * Purpose:  To delete an account from the database.
 * Returns:  return code indicating the status of execution.
 *******************************************************************/

void deleteAccount(
     handle_t h,
     accountInfo *data, /* data send by the client */
     unsigned32 *type,       /* type of account, i.e. WSM or WSR */
     unsigned32  *status /* return code sent to the client */
```

```
)

{
      int retcode;    /* Return code from the sendMulticast function */
      char resultsend[256]; /* Return string from the sendMulticast function */
      Message_t msend;    /* Message structure sent on the multicast address */
      char msg[256];
      char SQLstat[512];
         char transactionid[64];
      char busid[64];
      time_t calltime;

      time(&calltime);

     /*sprintf(msg,"deleteAccount called  for user %s\n",data->name);
      tmsg(msg);*/

      memset(resultsend,'\0',256);
      memset(transactionid,'\0',64);


         getuuid(busid,data->name);

      sprintf(transactionid,"%d%s",calltime,busid);

      memset ((void *)&msend, 0, sizeof(Message_t));

      memset(msend.msg,'\0',MaxMsg);

      msend.type=ntohl(UPDATE); /* Set the type of the multicast message */
      msend.App=htonl(*type);  /* Which application */

      sprintf(msend.msg,"%s,",transactionid);    /* delete account id */
         sprintf(SQLstat,"BEGIN ");
      strcat(msend.msg,SQLstat);


      /* Prepare the SQL query */
         if(*type==WSR)
         {
                  sprintf(SQLstat,"DELETE FROM %s WHERE
%s='%s';",WSR_AUTH,WSR_AUTH_NAME,data->name);
               strcat(msend.msg,SQLstat);
         }
      if(*type==WSM)
      {
                  sprintf(SQLstat,"DELETE FROM %s WHERE
%s='%s';",WSM_AUTH,WSM_AUTH_NAME,data->name);
            strcat(msend.msg,SQLstat);
      }
```

```
        sprintf(SQLstat,"END;");
        strcat(msend.msg,SQLstat);
        if(debug)printf("In deleteAccount. query *%s*\n",msend.msg);

        /* call the function to multicast the message to the listeners */
        if(sendMulticast(&msend,resultsend))
        {
               *status=1;    /* set the status code to indicate error */
        }

        else
        {
              *status=0;    /*set the status code to indicate successful deletion of the account */
        }

        if(debug) printf("Exiting deleteAccount with status %d\n",*status);

        /*sprintf(msg,"deleteAccount for user %s exited with status %d and result string %s\n",data-
>name,*status,result);
        tmsg(msg);*/

        return;
}

void resetAccount(
              handle_t  h,       /* DCE communication handle */
           accountInfo     *data,
        unsigned32 *type,       /* type of account, i.e. WSM or WSR */
        unsigned32  *status /* return code sent to the client */
            )
{
            int retcode;    /* Return code from the sendMulticast function */
        char resultsend[256]; /* Return string from the sendMulticast function */
        Message_t msend;    /* Message structure sent on the multicast address */
        char msg[256];
        char SQLstat[512];
            char transactionid[64];
        char busid[64];
        time_t calltime;

        time(&calltime);

       /*sprintf(msg,"resetAccount called  for user %s\n",data->name);
        tmsg(msg);*/

        memset(resultsend,'\0',256);
             memset(transactionid,'\0',64);


            getuuid(busid,data->name);
```

```
	sprintf(transactionid,"%d%s",calltime,busid);

	memset ((void *)&msend, 0, sizeof(Message_t));

	memset(msend.msg,'\0',MaxMsg);

	msend.type=ntohl(UPDATE); /* Set the type of the multicast message */
	msend.App=htonl(*type);  /* Which application */



	/* Prepare the SQL query */
	if(*type==WSR)
	{
	    sprintf(msend.msg,"%s,UPDATE %s SET ",transactionid,WSR_AUTH);
	              sprintf(SQLstat,"%s = '%s'",WSR_AUTH_WORD,data->word);
	      strcat(msend.msg,SQLstat);
	              sprintf(SQLstat," WHERE %s='%s'",WSR_AUTH_NAME,data->name);
	      strcat(msend.msg,SQLstat);
	    }
	if(*type==WSM)
	{
	    sprintf(msend.msg,"%s,UPDATE %s SET ",transactionid,WSM_AUTH);
	      sprintf(SQLstat,"%s = '%s'",WSM_AUTH_WORD,data->word);
	      strcat(msend.msg,SQLstat);
	              sprintf(SQLstat," WHERE %s='%s'",WSM_AUTH_NAME,data->name);
	      strcat(msend.msg,SQLstat);
	    }


	if(debug)printf("In resetAccount. query *%s*\n",msend.msg);

	/* call the function to multicast the message to the listeners */
	if(sendMulticast(&msend,resultsend))
	{
	      *status=1;    /* set the status code to indicate error */
	}

	else
	{
	     *status=0;    /*set the status code to indicate successful deletion of the account */
	}

	if(debug) printf("Exiting resetAccount with status %d\n",*status);

	/*sprintf(msg,"resetAccount for user %s exited with status %d and result string %s\n",data-
>name,*status,result);
	tmsg(msg);*/

	    return;
}
```

```
void updateAccount(
            handle_t  h,       /* DCE communication handle */
          accountInfo     *data,
              idl_char     *busid,
          idl_char  *result
       )
{


           Message_t msend;
           char SQLstat[256];
           int updateflag=0;
           char resultsend[256];
           char msg[256];
           char transactionid[64];
           time_t calltime;

       time(&calltime);

           /*sprintf(msg,"updateAccount called by user %s\n",data->name);
       tmsg(msg);*/

            memset(transactionid,'\0',64);



       sprintf(transactionid,"%d%s",calltime,busid);

           msend.type=htonl(UPDATE);
       msend.App=htonl(WSM);
           sprintf(msend.msg,"%s,UPDATE %s SET ",transactionid,WSR_AUTH);

       if(strlen(data->word)!=0)
       {
             sprintf(SQLstat,"%s = '%s'",WSR_AUTH_WORD,data->word);
             strcat(msend.msg,SQLstat);
             updateflag=1;
       }

           if(strlen(data->address)!=0)
       {
                     if(updateflag)strcat(msend.msg,", ");
             sprintf(SQLstat,"%s = '%s'",WSR_AUTH_ADD,data->address);
             strcat(msend.msg,SQLstat);
             updateflag=1;
       }

           if(strlen(data->phoneNo)!=0)
       {
```

```
                if(updateflag)strcat(msend.msg,", ");
            sprintf(SQLstat,"%s = '%s'",WSR_AUTH_PHONE,data->phoneNo);
            strcat(msend.msg,SQLstat);
            updateflag=1;
        }

          if(strlen(data->email)!=0)
        {
                if(updateflag)strcat(msend.msg,", ");
            sprintf(SQLstat,"%s = '%s'",WSR_AUTH_EMAIL,data->email);
            strcat(msend.msg,SQLstat);
            updateflag=1;
        }

          sprintf(SQLstat," WHERE %s='%s'",WSR_AUTH_BUSID,busid);
            strcat(msend.msg,SQLstat);

          if(debug)printf("In updateAccount. query *%s*\n",msend.msg);

          if(updateflag)
        {
            if(sendMulticast(&msend,resultsend))
            {
                strcpy(result,"Error in updating data. Contact administrator");
            }

            else
            {
                if(!strcmp(resultsend,"OK"))
                {
                    sprintf(result,"Data updated for the given account");
                }
                else
                {
                    strcpy(result,"Error in updating data. Contact administrator");
                }
            }

        }

          if(debug) printf("Exiting updateAccount with result string %s\n",result);

        /*sprintf(msg,"updateAccount for user %s exited with result string %s\n",data-
>name,result);
        tmsg(msg);*/

          return;
}

void updateServiceInfo(
            handle_t  h,       /* DCE communication handle */
        idl_char *serviceName, /* Name of the Service*/
        serviceInfo     *data,
            idl_char *busid,
          idl_char *result
        )
{
            Message_t msend;
        char SQLstat[256];
        int st,updateflag=0,i;
        char resultsend[256];
            char serviceid[64];
            char **columns[1];      /* Columns to be populated by the query  */
        int noColumns=1;       /* No of columns to be populated by the query */

            char transactionid[64];
             char msg[256];

            time_t calltime;

        time(&calltime);

        /*sprintf(msg,"updateServiceInfo called by user with busid %s\n",busid);
        tmsg(msg);*/

            memset(serviceid,'\0',64);

            memset(transactionid,'\0',64);

            /* Prepare a query to see whether or not a given service exists for the given client */
              sprintf(SQLstat,"select B.%s from %s A, %s B, %s C where A.%s = '%s' AND B.%s =
'%s' AND B.%s =
C.%s",WSM_SERVICE_SERVICEID,WSR_AUTH,WSM_SERVICE,WSR_BUSINESS,WSR_
AUTH_BUSID,busid,WSM_SERVICE_NAME,serviceName,WSM_SERVICE_SERVICEID,W
SR_BUSINESS_SERVICEID);

        if(debug)printf("In updateServiceInfo Query %s\n",SQLstat);

            for(i=0;i<noColumns;++i)
        {
              columns[i]=(char **)malloc(sizeof(char**));

        }

        /* call the function to query the database about the existence of the given service */
        st = fetch(SQLstat,columns,noColumns);


            if(st==0 || st==-1)
              {

                    if(st==-1)
                        strcpy(result,"ERROR: Contact Administrator");
                    else  strcpy(result,"ERROR: Service Does not exist with the given name");

                    for(i=0;i<noColumns;++i)
                    {
                        free(*(columns[i]));
                    }

                    return;
              }
               /* if the service is found, get the service id */
                    strcpy(serviceid,strtok(*(columns[0]),","));


        sprintf(transactionid,"%d%s",calltime,serviceid);

            for(i=0;i<noColumns;++i)
        {
              free(*(columns[i]));
        }


        msend.type=htonl(UPDATE);
        msend.App=htonl(WSM);
        sprintf(msend.msg,"%s,UPDATE %s SET ",transactionid,WSR_SERVICE);

        if(strlen(data->name)!=0)
        {
              sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_NAME,data->name);
              strcat(msend.msg,SQLstat);
              updateflag=1;
        }

        if(strlen(data->description)!=0)
        {
              if(updateflag)strcat(msend.msg,", ");
              sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_DESCRP,data->description);
              strcat(msend.msg,SQLstat);
              updateflag=1;
        }

        if(strlen(data->keywords)!=0)
        {
              if(updateflag)strcat(msend.msg,", ");
              sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_KEYWORDS,data->keywords);
              strcat(msend.msg,SQLstat);
              updateflag=1;
        }

        if(strlen(data->DocumentUrl)!=0)
        {
              if(updateflag)strcat(msend.msg,", ");
              sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_DOCURL,data->DocumentUrl);
              strcat(msend.msg,SQLstat);
              updateflag=1;
        }

            sprintf(SQLstat," WHERE %s='%s'",WSR_SERVICE_SERVICEID,serviceid);
              strcat(msend.msg,SQLstat);

        if(debug)printf("in updateServiceInfo Query *%s*\n",msend.msg);

            if(updateflag)
        {
              if(sendMulticast(&msend,resultsend))
              {
                    strcpy(result,"Error in updating data. Contact administrator");
              }

              else
              {
                    if(!strcmp(resultsend,"OK"))
                    {
                        sprintf(result,"Data updated for the given service");
                    }
                    else
                    {
                        strcpy(result,"Error in updating data. Contact administrator");
                    }
              }

        }

            if(debug) printf("Exiting updateServiceInfo with result string %s\n",result);

        /*sprintf(msg,"updateServiceInfo for user with busid %s exited with result string
%s\n",busid,result);
        tmsg(msg);*/

        return;


}

/*************************************************************************
 * Remote stopStatusService function in the DCEVEC Server Interface
 *
 * Purpose:  To stop or get status of a webservice.
 * Function: Forks a child process for each generic path it has to query, waits for the child
processes to finish execution and reads
 *             the results written by the child processes from the respective file(s).If request is to stop
the service, it forms a query
```

```
 *             to remove the entry from the database and multicasts the query.
 * Returns:  return code indicating the status of execution.
 *         return 0 if the query is successfully executed.
 *         return 1 if error and  send the error String back to the client.
 *             result is copied to the result string sent by the calling function.
 ***********************************************************************/
int stopStatusService(idl_char *serviceName,  /* Serive name */
                              char *busid,          /* unique busid of the client */
                              char *WSRid,          /* service id */
                              char **pathGeneric,  /* array of generic paths */
                              int noGenPath,       /* no of generic paths */
                              char **result,      /* result to be sent back to the calling function */
                              int type)               /*type of operation i.e STOP or STATUS of a web-
service */
{

       char resultfrom[256],
           filename[128];  /* File created by the child processes */
        Message_t msend; /* Message structure to be multicasted */
           char *temp;
       int *pid,*status; /* Process id's and exit status of the child processes */
       int len,i,j,statusret=0;
       char SQLstat[512];  /* Form query */
           FILE *fp;
           char msg[256];
           char transactionid[64];
           time_t calltime;
           char **columns[1];
           int noColumns,st;
           char serviceUrl[128];
           char query[256];
       time(&calltime);
       clock_t clock(),startcputime,endcputime;
           time_t  startexectime,endexectime;
            memset(transactionid,'\0',64);
            memset(serviceUrl,'\0',128);


       sprintf(transactionid,"%d%s",calltime,busid);

           if(debug)printf("stopStatusService called with service *%s* type %d and no paths
%d\n",WSRid,type,noGenPath);

            pid=(int*)malloc(sizeof(int)*noGenPath); /* Process ids of the child processes */
       status=(int*)malloc(sizeof(int)*noGenPath); /* exit status of the child processes */

       len=0;
           memset ((void *)&msend, 0, sizeof(Message_t));
       memset(msend.msg,'\0',MaxMsg);
           memset(resultfrom,'\0',256);
           sprintf(SQLstat,"BEGIN ");
           sprintf(msend.msg,"%s,",transactionid);
           strcat(msend.msg,SQLstat);

           time(&startexectime);
       startcputime=clock();
           /* for the number of generic paths passed to this function */
       for(i=0;i<noGenPath;++i)
       {
            memset(resultfrom,'\0',128);

                    /* create child proces for each generic path */
                    if((pid[i]=fork())==0)
            {
            if(debug)printf("Child created with pid %d\n",getpid());

                     if(type==STATUS)
                    {
                             noColumns=1;

                         for(j=0;j<noColumns;++j)
                    {
                              columns[j]=(char **)malloc(sizeof(char**));

                         }

                              sprintf(query,"SELECT %s from %s where %s='%s' And
%s='%s'",WSM_REPLICA_SERVICEURL,WSM_REPLICA,WSM_REPLICA_SERVICEID,
WSRid,WSM_REPLICA_GENERICURL,pathGeneric[i]);
                              if(debug)printf("Fetching serviceUrl with wsrid *%s* Calling fetch with
query *%s*\n",WSRid,query);


                              st = fetch(query,columns,noColumns);

                              strcpy(serviceUrl,strtok(*(columns[0]),","));

                              callGenericService(method[type],WSRid,serviceUrl,resultfrom);

                              for(j=0;j<noColumns;++j)
                       {
                        free(*(columns[j]));
                       }

                    }
                    else
                    {
                    /* call the method to contact the generic service and perform the required
operation */
                    callGenericService(method[type],WSRid,pathGeneric[i],resultfrom);
```

```
                }

                /* write the result to the file */
            sprintf(filename,"%d",getpid());
            fp = fopen(filename,"w");
            fprintf(fp,"%s",resultfrom);
            fclose(fp);


            if(debug)printf("Child %d exited with result *%s*\n",getpid(),resultfrom);
            exit(1);
            }

    }

          for(i=0;i<noGenPath;++i)
          {
            if(debug)printf("Waiting for process %d\n",pid[i]);
                /* wait for the child processes */
            waitpid(pid[i],&status[i],0);

                /* read the result from the file */
            sprintf(filename,"%d",pid[i]);
            printf("file *%s*\n",filename);
            fp = fopen(filename,"r");
            fgets(resultfrom,128,fp);
                fclose(fp);
            unlink(filename);
            len = strlen(resultfrom) + len +1;

                if(debug) printf("result read from file by %d is *%s*\n",pid[i],resultfrom);
                /* form the query to delete the entry from the database if the request is to stop the
webservice */
                if(type==STOP)
            {
                sprintf(SQLstat,"DELETE FROM %s WHERE %s= (SELECT %s FROM %s
WHERE %s = '%s' AND %s = '%s');
",WSR_ACCESSPOINTS,WSR_ACCESSPOINTS_URL,WSM_REPLICA_SERVICEURL,WS
M_REPLICA,WSM_REPLICA_GENERICURL,pathGeneric[i],WSM_REPLICA_SERVICEID,
WSRid);
            strcat(msend.msg,SQLstat);
                sprintf(SQLstat,"DELETE FROM %s WHERE %s='%s' AND %s = '%s';
",WSM_REPLICA,WSM_REPLICA_GENERICURL,pathGeneric[i],WSM_REPLICA_SERVI
CEID,WSRid);
            strcat(msend.msg,SQLstat);

               }

                /* form the result string to be sent to the calling function */
            *result=(char *)realloc((void*)*result,sizeof(char)*len);
            strcat(*result,resultfrom);
            strcat(*result,",");
         }

          time(&endexectime);
          endcputime=clock();

          printf("Time taken to contact the genericSoapServers cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

                    sprintf(SQLstat,"END;");
                    strcat(msend.msg,SQLstat);
                    if(debug)printf("Query *%s*\n",msend.msg);
         msend.type=htonl(UPDATE);
          msend.App=htonl(WSM);  /* Which application */
          memset(resultfrom,'\0',128);


          /* send the multicast message to update the database if the request is to stop the
webservice */
          if(type==STOP)
          {
            if(sendMulticast(&msend,resultfrom))
         {
            statusret=1;
         }

         else
          {
            if(!strcmp(resultfrom,"OK"))
            {
                  statusret=0;
            }
          else  statusret=1;

              }

      }

          if(debug) printf("stopstatusService exited with status %d and string
*%s*\n",statusret,*result);

          free(pid);
          free(status);
      return statusret;

}

/*************************************************************************
 * Remote createStartService function in the DCEVEC Server Interface
 *
 * Purpose:  To create a group of webservices or start a webservice for an already existing group.
 * Function: Forks a child process for each generic path , waits for the child processes to finish
execution and reads
 *          the results written by the child processes from the respective file(s). It forms a query
 *          to add the entry to the database for the service started and multicasts the query.
 * Returns:  return code indicating the status of execution.
 *          return 0 if the query is successfully executed.
 *          return 1 if error and  send the error String back to the client.
 *          result is copied to the result string sent by the calling function.
 ***********************************************************************/
int createStartService(idl_char *serviceName, /* Service name */
                              serviceInfo *serviceDet, /* Details about the service */
                              char *busid,      /* unique business id of the client */
                              char *WSRid,    /* service id */
                              char **pathGeneric, /* paths of the generic webservice */
                              char **pathStandby, /* paths of the standby servers */
                              int noGenPath,      /* number of generic paths */
                              int noStandbyPath,  /* number of standby servers */
                              char **result,    /* result sent to the calling function */
                              int type)    /* Type of request i.e CREATE a webservice group or
START a webservice.*/
{

          char resultfrom[256]
          ,filename[128];  /* File created by the child processes */
      char WSR_service[128];
          char *temp;
      int *pid,*status; /* Process id's and exit status of the child processes */
      int len,i,statusret=0;
      FILE *fp;
          char SQLstat[512]; /* Form query */
           Message_t msend;        /* Message structure to be multicasted */
          char resultsend[256];
          char transactionid[64];
          int doupdate=0; /*updateflag to indicate whether database has to be updated or not */
          time_t calltime;
          clock_t clock(),startcputime,endcputime;
      time_t startexectime,endexectime;
          time(&calltime);

          pid=(int*)malloc(sizeof(int)*noGenPath); /* Process ids of the child processes */
      status=(int*)malloc(sizeof(int)*noGenPath); /* exit status of the child processes */
      len=0;

           memset(transactionid,'\0',64);

          if(debug)printf("createService called with service *%s* type %d and no paths
%d\n",WSRid,type,noGenPath);
          memset ((void *)&msend, 0, sizeof(Message_t));

          memset(msend.msg,'\0',MaxMsg);
          memset(resultfrom,'\0',256);
```

```
    sprintf(transactionid,"%d%s",calltime,WSRid);

        /* attach the service name to the WSRid */
        sprintf(WSR_service,"%s,%s",WSRid,serviceName);

        sprintf(SQLstat,"BEGIN ");
        sprintf(msend.msg,"%s,",transactionid);
        strcat(msend.msg,SQLstat);

        /* If a webservice group is being created, enter information about the webservice in the
respective tables */
        if(type==CREATE)
        {
            sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s');
",WSR_BUSINESS,busid,WSRid);
        strcat(msend.msg,SQLstat);
            sprintf(SQLstat,"INSERT INTO %s  VALUES ('%s','%s','%s','%s','%s');
",WSR_SERVICE,WSRid,serviceDet->name,serviceDet->description,serviceDet-
>keywords,serviceDet->DocumentUrl);
        strcat(msend.msg,SQLstat);
                sprintf(SQLstat,"INSERT INTO %s  VALUES ('%s','%s','%s');
",WSM_SERVICE,busid,serviceName,WSRid);
        strcat(msend.msg,SQLstat);

        }
        time(&startexectime);
        startcputime=clock();

        /* for the number of generic paths passed to this function */
        for(i=0;i<noGenPath;++i)
    {
         memset(resultfrom,'\0',128);

                /* create child proces for each generic path */
        if((pid[i]=fork())==0)
         {
                if(debug)printf("Child created with pid %d\n",getpid());
         /* call the method to contact the generic service and perform the required operation */
                callGenericService(method[type],WSR_service,pathGeneric[i],resultfrom);

                /* write the result to the file */
         sprintf(filename,"%d",getpid());
         fp = fopen(filename,"w");
         fprintf(fp,"%s",resultfrom);
         fclose(fp);

                 if(debug)printf("Child %d exited with result *%s*\n",getpid(),resultfrom);
         exit(1);
         }
```

```
      }

         for(i=0;i<noGenPath;++i)
         {

                   if(debug)printf("Waiting for process %d\n",pid[i]);
                   /* wait for the child processes */
           waitpid(pid[i],&status[i],0);
                   if(debug)printf("Process %d exited with status %d\n",pid[i],status[i]>>8);

                   /* read the result from the file */
            sprintf(filename,"%d",pid[i]);
            fp = fopen(filename,"r");

                            fgets(resultfrom,128,fp);
                            printf("File *%s*\n",resultfrom);
                   fclose(fp);
                   unlink(filename);
                   len = strlen(resultfrom) + len +1;

                            if(debug) printf("result read from file by %d is
*%s*\n",pid[i],resultfrom);

                   /* IF no error in starting the webservice */
                   if(strncmp(resultfrom,"ERROR",5))
                   {
                   sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s','%s',Sysdate);
",WSM_REPLICA,WSRid,pathGeneric[i],resultfrom);
          strcat(msend.msg,SQLstat);
                   sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s');
",WSR_ACCESSPOINTS,WSRid,resultfrom);
      strcat(msend.msg,SQLstat);

                            if(type==START)
                   {
                   sprintf(SQLstat,"DELETE FROM %s WHERE %s='%s' AND %s ='%s';
",WSM_STANDBY,WSM_STANDBY_GENERICURL,pathGeneric[i],WSM_STANDBY_SER
VICEID,WSRid);
          strcat(msend.msg,SQLstat);
                            }
                   doupdate=1;  /* Do the update to the database only if atleast one service has been
started */
                   }


                   /* form the result string to be sent to the calling function */

            *result=(char *)realloc((void*)*result,sizeof(char)*len);
            strcat(*result,resultfrom);
            strcat(*result,",");
         }
```

```
        time(&endexectime);
  endcputime=clock();

        printf("Time taken to contact the genericSoapServers cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

                /* if creating a web-service group enter the standby paths in the database */
                if(type==CREATE)
                {
                        for(i=0;i<noStandbyPath;++i)
                        {
                        sprintf(SQLstat,"INSERT INTO %s (%s,%s) VALUES ('%s','%s');
",WSM_STANDBY,WSM_STANDBY_SERVICEID,WSM_STANDBY_GENERICURL,WSRi
d,pathStandby[i]);
                        strcat(msend.msg,SQLstat);
                        }
                }
          sprintf(SQLstat,"END;");
        strcat(msend.msg,SQLstat);

        if(debug)printf("query *%s*\n",msend.msg);

    msend.type=htonl(UPDATE);
        msend.App=htonl(WSM);  /* Which application */

        if(doupdate)
        {
                if(sendMulticast(&msend,resultsend))
        {
                statusret=1;
        }

        else
        {
                if(!strcmp(resultsend,"OK"))
                {
               statusret=0;
                }
                else
                {
               statusret=1;
                }
        }
     }
        else
        {
                statusret=-1;
        }

        if(debug) printf("createService exited with status %d and string
*%s*\n",statusret,*result);
```

```
          free(pid);
       free(status);
          return statusret;


}

/*********************************************************************
 * Remote serviceAction function in the DCEVEC Server Interface
 *
 * Purpose:  To accept a request from the client to create a webservice group or start,stop and get
status of a webservice.
 * Function: To parse the data sent by the client and as per the request call the respective
functions to start,stop or get
 *             status of the webservice.
 * Returns:  return code indicating the status of execution.
 *         return 0 if the query is successfully executed to insert a row and also send an unique-id
back to the client.
 *         return 1 if error and  send the error String back to the client.
 *********************************************************************/
void serviceAction(
          handle_t h,
          idl_char *serviceName,  /* Name of the webservice */
          serviceInfo *serviceDet,  /* details about the service */
          idl_char *paths,        /* Urls of the generice web service to be called to execute the
request on the server */
          idl_char *standbypaths, /* Urls of the standby servers to be brought up in case of a
failure. */
       idl_char busid[64],     /* Unique-id of the client to ensure he has the authority to administer
the given service */
          unsigned32 *type,       /* type of request, i.e. create, start, stop or status */
          conarray *result ,       /* result sent back to the client */
          unsigned32 *status
          )
{
          int noGenPath=0, /* Number of generic paths sent by the client */
          j,i,
          noStandbyPath=0; /* Number of standby paths sent by the client */
          char **pathGeneric;    /* To generate an array of Strings of the Urls of the generic web-
services to be called */
          char **pathStandby;     /* To generate an arrya of Strings of the Urls of the standby
servers */
          char **resultclient;
          int  call,st;
          char *temp;
          char serviceId[64];     /* To hold the service id for the given service */
          char query[512];       /* query to be sent to the fetch function */
          char **columns[1];      /* Columns to be populated by the query  */
       int noColumns=1;       /* No of columns to be populated by the query */
          char msg[256];
```

```
        /*sprintf(msg,"ServiceAction called by user with business id *%s* with type
%d\n",busid,*type);
        tmsg(msg);*/

        memset(serviceId,'\0',64);
        resultclient=(char **)malloc(sizeof(char));
        /* Prepare a query to see whether or not a given service exists for the given client */
                sprintf(query,"select B.%s from %s A, %s B, %s C where A.%s = '%s' AND
B.%s = '%s' AND B.%s =
C.%s",WSM_SERVICE_SERVICEID,WSR_AUTH,WSM_SERVICE,WSR_BUSINESS,WSR_
AUTH_BUSID,busid,WSM_SERVICE_NAME,serviceName,WSM_SERVICE_SERVICEID,W
SR_BUSINESS_SERVICEID);

        if(debug)printf("In serviceAction with query *%s*\n",query);

    for(i=0;i<noColumns;++i)
     {
         columns[i]=(char **)malloc(sizeof(char**));

     }

        /* call the function to query the database about the existence of the given service */
        st = fetch(query,columns,noColumns);

        /* If the request is any of the three, the service should exist in the database. If it doesn't
return error*/
        if(*type==START || *type==STOP || *type == STATUS)
          {

                if(st==0 || st==-1)
                {

              *resultclient=(char *)realloc((void*)*resultclient,sizeof(char)*64);
                        if(st==-1)
                                strcpy(*resultclient,"ERROR: Contact Administrator");
                        else  strcpy(*resultclient,"ERROR: Service Does not exist with the given
name");
                        result->first=0;
                         result->size=strlen(*resultclient)+1;
                         strcpy(result->array,*resultclient);
                        free(*resultclient);
                        free(resultclient);

                        for(i=0;i<noColumns;++i)
                {
                        free(*(columns[i]));
                }
                        *status=-1;
                        if(debug)printf("Exiting seriveAction with status %d and result string
*%s*\n",*status,result->array);
```

```
                    /*sprintf(msg,"ServiceAction exited for busid %s with status %d result
string %s\n",busid,*status,result->array);
                    tmsg(msg);*/

                    return;
               }
                 /* if the service is found, get the service id */
                    strcpy(serviceId,strtok(*(columns[0]),","));

        }

        /* If the request is to create a new webservice group, the service should not exist for the
given name. If it does return error*/
        if(*type==CREATE)
        {

               /*See if the service has been created with the same servicename*/
               if(st==-1 || st>0)
          {
              *resultclient=(char *)realloc((void*)*resultclient,sizeof(char)*64);
              if(st==-1)
                    strcpy(*resultclient,"ERROR: Contact Administrator");
              else strcpy(*resultclient,"ERROR:Service already exists with this name");
                     result->first=0;
               result->size=strlen(*resultclient)+1;
               strcpy(result->array,*resultclient);
              free(*resultclient);
              free(resultclient);
                    for(i=0;i<noColumns;++i)
               {
                    free(*(columns[i]));
               }
                    *status=-1;
                 if(debug)printf("Exiting seriveAction with status %d and result string
*%s*\n",*status,result->array);
                    /*sprintf(msg,"ServiceAction exited for busid %s with status %d result
string %s\n",busid,*status,result->array);
                    tmsg(msg);*/
             return;
          }

               getuuid(serviceId,serviceName);

        }

        for(i=0;i<noColumns;++i)
     {
         free(*(columns[i]));
     }

        pathGeneric = (char **)malloc (sizeof(char **));
```

```
pathStandby = (char **)malloc (sizeof(char **));


    /* Parse the comma separated generic paths to form an array of Strings */
    if(strlen(paths)!=0)
{

      if((temp=strtok(paths,","))!=NULL)
      {
            pathGeneric[noGenPath]=(char *)malloc(sizeof(char*));
           pathGeneric[noGenPath]=(char *)malloc(sizeof(char)*(strlen(temp)+1));
            strcpy(pathGeneric[noGenPath],temp);
            noGenPath++;
      }

      while((temp=strtok(NULL,","))!=NULL)
      {
            pathGeneric[noGenPath]=(char *)malloc(sizeof(char*));
            pathGeneric[noGenPath]=(char *)malloc(sizeof(char)*(strlen(temp)+1));
            strcpy(pathGeneric[noGenPath],temp);
            noGenPath++;
      }

}


    /* Parse the comma separated standby paths to form an array of Strings */
    if(strlen(standbypaths)!=0)
{

      if((temp=strtok(standbypaths,","))!=NULL)
      {
            pathStandby[noStandbyPath]=(char *)malloc(sizeof(char*));
           pathStandby[noStandbyPath]=(char *)malloc(sizeof(char)*(strlen(temp)+1));
            strcpy(pathStandby[noStandbyPath],temp);
            noStandbyPath++;
      }

      while((temp=strtok(NULL,","))!=NULL)
      {
            pathStandby[noStandbyPath]=(char *)malloc(sizeof(char*));
            pathStandby[noStandbyPath]=(char *)malloc(sizeof(char)*(strlen(temp)+1));
            strcpy(pathStandby[noStandbyPath],temp);
            noStandbyPath++;
      }

}


    /* Call the respective function based on the request */
    if(*type==CREATE|| *type==START)
```

```
            call =
createStartService(serviceName,serviceDet,busid,serviceId,pathGeneric,pathStandby,noGenPath,
noStandbyPath,resultclient,*type);
            else call =
stopStatusService(serviceName,busid,serviceId,pathGeneric,noGenPath,resultclient,*type);

             for(j=0;j<noGenPath;++j)
        {
                    free(pathGeneric[j]);
        }

            for(j=0;j<noStandbyPath;++j)
        {
            free(pathStandby[j]);
        }

            /* if the return val is zero copy the result to sent to the client */
            if(call==0)
            {
                    result->first=0;
                    result->size=strlen(*resultclient)+1;
            strcpy(result->array,*resultclient);
                    *status=0;
            }
            else if(call ==1)    /* else indicate error to the client */
            {
                    result->first=0;
              strcpy(result->array,*resultclient);
              result->size=strlen(result->array)+1;
                    *status =1;
            }
            else if (call==-1)
            {
                    result->first=0;
              strcpy(result->array,*resultclient);
              result->size=strlen(result->array)+1;
              *status =-2;
            }

            if(debug)printf("Exiting seriveAction with status %d and result string
*%s*\n",*status,result->array);

            /*sprintf(msg,"seriveAction exited for busid %s with status %d and result string
*%s*\n",busid,*status,result->array);
            tmsg(msg);*/

            free(pathGeneric);
        free(pathStandby);
            free(resultclient);
            return;
}
```

```
/***********************************************************************
 * Remote updateStandby function in the DCEVEC Server Interface
 *
 * Purpose:  To accept a request from the client to add standby servers for a given service.
 * Function: To parse the data sent by the client, prepare a query to add the standby servers to the
database for the given service.
 * Returns:  return string indicating the status of execution.
 ***********************************************************************/

void updateServers(
  /* [in]   */        handle_t h,
          unsigned32 *type,       /* type of request, i.e. delete or add*/
   /*[in,string,ptr]*/   idl_char *serviceName,  /* Name of the Service */
  /* [in,string,ptr]*/   idl_char *standbypaths, /* Comman separated standby paths */
  /* [in,string,ptr]*/   idl_char *busid,       /*Unique-id for the given client */
  /* [out,string] */     idl_char *rstatus       /* String sent back to the client */
)
{

          char SQLstat[256];
        Message_t msend; /* Message sturcture to be multicasted to the listeners */
          char **pathStandby, /* To form an array of Strings for the standby paths */
          *temp;
          int st,j,i,
          updateflag=0;    /* To indicate atleast one standby path has to be added*/
          int noStandbyPath=0;    /* Number of standby paths */
          char  WSRid[64];           /* Service id for the service */
          char resultsend[256];
          char **columns[1];   /* columns to be populated by the query */
          int noColumns=1;           /* Number of columns to be populated by the query */
          char msg[256];
          char transactionid[64];
          time_t calltime;

       time(&calltime);

          memset(resultsend,'\0',256);

          pathStandby = (char **)malloc (sizeof(char **));

          /*sprintf(msg,"updateServers called by user with business id %s and service name
%s\n",busid,serviceName);
          tmsg(msg);*/

           memset(transactionid,'\0',64);


       sprintf(transactionid,"%d%s",calltime,busid);

          if(strncmp(serviceName,"GENERIC",7))
        {
                /* Prepare a query to see whether or not a given service exists for the given client
*/
                sprintf(SQLstat,"select B.%s from %s B, %s C where C.%s = '%s' AND B.%s =
'%s' AND B.%s =
C.%s",WSM_SERVICE_SERVICEID,WSM_SERVICE,WSR_BUSINESS,WSR_BUSINESS_
BUSID,busid,WSM_SERVICE_NAME,serviceName,WSM_SERVICE_SERVICEID,WSR_BU
SINESS_SERVICEID);

                for(i=0;i<noColumns;++i)
        {
                columns[i]=(char **)malloc(sizeof(char**));

        }

                if(debug)printf("In UpdateStandby with query *%s*\n",SQLstat);

                /* call the function to query the database about the existence of the given service
*/
                st = fetch(SQLstat,columns,noColumns);


        /* check to see if the service exists and is registered  by the client. If it doesn't then exit*/
          if(st==0 || st==-1)
          {

               if(st==-1)
                    strcpy(rstatus,"ERROR: Contact Administrator");
               else  strcpy(rstatus,"ERROR: Service Does not exist with the given name");

               for(i=0;i<noColumns;++i)
               {
                    free(*(columns[i]));
               }

                         if(debug)printf("UpdateStandby exited with rstatus *%s*\n",rstatus);

                /*sprintf(msg,"UpdateStandby exited with rstatus *%s*\n",rstatus);
                tmsg(msg);*/

               return;
          }


                         /* If service exists, get the service id */
            strcpy(WSRid,strtok(*(columns[0]),","));


                 for(i=0;i<noColumns;++i)
               {
                    free(*(columns[i]));
                }
            }

            if(strlen(standbypaths)!=0)
        {

              if((temp=strtok(standbypaths,","))!=NULL)
              {
                    pathStandby[noStandbyPath]=(char *)malloc(sizeof(char*));
                   pathStandby[noStandbyPath]=(char *)malloc(sizeof(char)*(strlen(temp)+1));
                    strcpy(pathStandby[noStandbyPath],temp);
                    noStandbyPath++;
              }

              while((temp=strtok(NULL,","))!=NULL)
              {
                    pathStandby[noStandbyPath]=(char *)malloc(sizeof(char*));
                    pathStandby[noStandbyPath]=(char *)malloc(sizeof(char)*(strlen(temp)+1));
                    strcpy(pathStandby[noStandbyPath],temp);
                    noStandbyPath++;
              }

        }

            /* Form the query to insert the standby paths in the database */
             sprintf(SQLstat,"BEGIN ");
        sprintf(msend.msg,"%s,",transactionid);
            strcat(msend.msg,SQLstat);

            if(debug)printf("type = %d number paths %d\n",*type,noStandbyPath);
            for(i=0;i<noStandbyPath;++i)
              {
                              if(*type==1)
                              {
                              if(!strncmp(serviceName,"GENERIC",7))
                                      {
                                      sprintf(SQLstat,"INSERT INTO %s  VALUES ('%s','%s');
",WSM_GENERIC,busid,pathStandby[i]);
                                      }
                                      else
                                      {
                                      sprintf(SQLstat,"INSERT INTO %s (%s,%s) VALUES
('%s','%s');
",WSM_STANDBY,WSM_STANDBY_SERVICEID,WSM_STANDBY_GENERICURL,WSRi
d,pathStandby[i]);
                              }
                              }
                              else if (*type==2)
                              {
                              if(!strncmp(serviceName,"GENERIC",7))
                          {
```

```
                                        sprintf(SQLstat,"DELETE FROM %s where %s = '%s' AND %s
= '%s';
",WSM_GENERIC,WSM_GENERIC_BUSID,busid,WSM_GENERIC_GENERICURL,pathSta
ndby[i]);

                              }
                              else
                              {
                                        sprintf(SQLstat,"DELETE FROM %s where %s = '%s' AND %s
= '%s';
",WSM_STANDBY,WSM_STANDBY_SERVICEID,WSRid,WSM_STANDBY_GENERICUR
L,pathStandby[i]);
                                        }
                                 }
                                 strcat(msend.msg,SQLstat);
                                 updateflag=1;
             }

         sprintf(SQLstat,"END;");
        strcat(msend.msg,SQLstat);
        if(debug)printf("query *%s*\n",msend.msg);

          /* set the type of message sent on the multicast address */
          msend.type=htonl(UPDATE);
          msend.App=htonl(WSM);  /* Which application */

          if(updateflag)
          {
                 if(sendMulticast(&msend,resultsend))
          {
                strcpy(rstatus,"Error in updating the list of servers");
          }

          else
          {
                if(!strcmp(resultsend,"OK"))
                {
                strcpy(rstatus,"List of servers updated successfully");
                }
                else
                {
                      strcpy(rstatus,"Error in updating the list of servers");
                }
          }
          }
          else
          {
                strcpy(rstatus,"No servers provided");
          }

          if(debug)printf("UpdateServers exited with rstatus *%s*\n",rstatus);
        /*sprintf(msg,"UpdateServers exited with rstatus *%s*\n",rstatus);
        tmsg(msg);*/
        return;

}

/***********************************************************************
 * Remote deleteService function in the DCEVEC Server Interface
 *
 * Purpose:  To accept a request from the client to delete a given service.
 * Returns:  return string indicating the status of execution.
 ***********************************************************************/

void deleteService(
  /* [in]   */        handle_t h,
   /*[in,string,ptr]*/   idl_char *serviceName,  /* Name of the Service */
  /* [in,string,ptr]*/   idl_char *busid,        /*Unique-id for the given client */
  /* [out,string] */     idl_char *rstatus       /* String sent back to the client */
)
{
        char SQLstat[256];
     Message_t msend;       /* Message sturcture to be multicasted to the listeners */
        int st,j,i;
         char  WSRid[64];        /* Service id for the service */
     char resultsend[256];
     char **columns[1];   /* columns to be populated by the query */
     int noColumns=1;        /* Number of columns to be populated by the query */
     char msg[256];
        char transactionid[64];
        time_t calltime;

     time(&calltime);

         memset(resultsend,'\0',256);

         memset(transactionid,'\0',64);


     sprintf(transactionid,"%d%s",calltime,busid);

        /*sprintf(msg,"deleteService called by user with business id %s and service name
%s\n",busid,serviceName);
     tmsg(msg);*/

        /* Prepare a query to see whether or not a given service exists for the given client */
     sprintf(SQLstat,"select B.%s from %s A, %s B, %s C where A.%s = '%s' AND B.%s = '%s'
AND B.%s =
C.%s",WSM_SERVICE_SERVICEID,WSR_AUTH,WSM_SERVICE,WSR_BUSINESS,WSR_
AUTH_BUSID,busid,WSM_SERVICE_NAME,serviceName,WSM_SERVICE_SERVICEID,W
SR_BUSINESS_SERVICEID);
```

```
        for(i=0;i<noColumns;++i)
        {
              columns[i]=(char **)malloc(sizeof(char**));

        }

        if(debug)printf("In deleteService with query *%s*\n",SQLstat);

       /* call the function to query the database about the existence of the given service */
        st = fetch(SQLstat,columns,noColumns);

            /* check to see if the service exists and is registered  by the client. If it doesn't then exit*/
              if(st==0 || st==-1)
              {

                    if(st==-1)
                          strcpy(rstatus,"ERROR: Contact Administrator");
                    else  strcpy(rstatus,"ERROR: Service Does not exist with the given name");

                    for(i=0;i<noColumns;++i)
                    {
                          free(*(columns[i]));
                    }

                    if(debug)printf("deleteService exited with rstatus *%s*\n",rstatus);

                    /*sprintf(msg,"deleteService exited with rstatus *%s*\n",rstatus);
                    tmsg(msg);*/

                    return;
              }


                    /* If service exists, get the service id */
                 strcpy(WSRid,strtok(*(columns[0]),","));


        for(i=0;i<noColumns;++i)
                    {
                          free(*(columns[i]));
                    }


            /* Prepare a query to see whether any replicas are running for the given service */
        sprintf(SQLstat,"select %s from %s where %s =
'%s'",WSM_REPLICA_SERVICEURL,WSM_REPLICA,WSM_REPLICA_SERVICEID,WSRi
d);

            noColumns=1;
        for(i=0;i<noColumns;++i)
        {
                columns[i]=(char **)malloc(sizeof(char**));

        }

        if(debug)printf("In deleteService with query *%s*\n",SQLstat);

       /* call the function to query the database about the existence of the replicas of the service */
         st = fetch(SQLstat,columns,noColumns);

        /* check to see if any replicas are running. IF running exit*/
                if(st>0 || st==-1)
                {

                        if(st==-1)
                                strcpy(rstatus,"ERROR: Contact Administrator");
                        else  strcpy(rstatus,"ERROR: Service still running on servers. Stop the service first
and then delete the service");

                        for(i=0;i<noColumns;++i)
                        {
                                free(*(columns[i]));
                        }

                        if(debug)printf("deleteService exited with rstatus *%s*\n",rstatus);

                        /*sprintf(msg,"deleteService exited with rstatus *%s*\n",rstatus);
                        tmsg(msg);*/

                        return;
                }

             for(i=0;i<noColumns;++i)
                        {
                                free(*(columns[i]));
                        }

             /* Form the query to delete the service group from the database */
         sprintf(SQLstat,"BEGIN ");
        sprintf(msend.msg,"%s,",transactionid);
        strcat(msend.msg,SQLstat);

             sprintf(SQLstat,"DELETE FROM %s where %s = '%s';
",WSR_BUSINESS,WSR_BUSINESS_SERVICEID,WSRid);
             strcat(msend.msg,SQLstat);

             sprintf(SQLstat,"DELETE FROM %s where %s = '%s';
",WSM_SERVICE,WSM_SERVICE_SERVICEID,WSRid);
        strcat(msend.msg,SQLstat);
```

```
         sprintf(SQLstat,"END;");
      strcat(msend.msg,SQLstat);
      if(debug)printf("query *%s*\n",msend.msg);

      /* set the type of message sent on the multicast address */
      msend.type=htonl(UPDATE);
      msend.App=htonl(WSM);  /* Which application */

            if(sendMulticast(&msend,resultsend))
            {
                  strcpy(rstatus,"Error in deleting service");
            }

            else
            {
                  if(!strcmp(resultsend,"OK"))
                  {
                        strcpy(rstatus,"Service deleted from the database");
                  }
                  else
                  {
                        strcpy(rstatus,"Error in deleting service");
                  }
            }

      if(debug)printf("deleteService exited with rstatus *%s*\n",rstatus);

         /*sprintf(msg,"deleteService exited with rstatus *%s*\n",rstatus);
      tmsg(msg);*/
      return;

}
/***********************************************************************
 * Remote getBusidInfo function in the DCEVEC Server Interface
 *
 * Purpose:  To get a list of all the services running under a given unique business id .
 * Function: To prepare a query to query the database for the all the services running for a given
unique business id.
 * Returns:  return string of the service names found.
 *                    return string of the respective service ids found.
 *                    return code to indicate the status.
***********************************************************************/

void getBusidInfo(
           handle_t  h,       /* DCE communication handle */
          idl_char *Busid, /* Busid of the Client */
          accountInfo     *data,
             conarray *serviceName, /* Comma separated service names to be sent to the client */
           conarray *serviceId,       /* Respective Comma separated service ids to be send to the
client */
            unsigned32 *status        /* return code */
```

```
)
{

        char query[512];
    char result[64];
    int st,i;
    char **columns[3], /* Columns to be populated by the query */
        *temp;
    int noColumns;   /* Number of Columns to be populated by the query */
        char msg[256];

        /*sprintf(msg,"getBusidInfo called by user with busid %s\n",Busid);
        tmsg(msg);*/

         data->name = (char*)malloc(sizeof(char)*64);
                data->word = (char*)malloc(sizeof(char)*64);
                data->address = (char*)malloc(sizeof(char)*256);
                data->phoneNo = (char*)malloc(sizeof(char)*64);
                data->email = (char*)malloc(sizeof(char)*64);
        /* Form the query to get the account info */
    sprintf(query,"SELECT %s,%s,%s FROM %s WHERE %s
='%s'",WSR_AUTH_ADD,WSR_AUTH_EMAIL,WSR_AUTH_PHONE,WSR_AUTH,WSR_A
UTH_BUSID,Busid);

    if(debug)printf("In getbusidInfo with query *%s*\n",query);

        noColumns=3;
       for(i=0;i<noColumns;++i)
    {
          columns[i]=(char **)malloc(sizeof(char**));

    }


    /* call the function to execute the query and return the results */
    st = fetch(query,columns,noColumns);

    /* If error in querying the database */
    if(st==-1 || st==0)
    {
          strcpy(result,"ERROR:Contact Administrator");

                 strcpy(data->name,result);
                 strcpy(data->word,"");
                 strcpy(data->address,"");
                 strcpy(data->email,"");
                 strcpy(data->phoneNo,"");
          serviceName->first=0;
          serviceName->size=strlen(result)+1;
          strcpy(serviceName->array,result);
           serviceId->first=0;
```

```
            serviceId->size=strlen(result)+1;
            strcpy(serviceId->array,result);
                    *status=2;
                     for(i=0;i<noColumns;++i)
          {
                    free(*(columns[i]));
          }

                    return;
          }
      else     /* Else copy the names and ids to be sent to the client */
      {
                    if(debug)
                    {
                              printf("*%s* *%s*
*%s*\n",*(columns[0]),*(columns[1]),*(columns[2]));
                    }
                    strcpy(data->name,"");
            strcpy(data->word,"");

                    if((temp=strtok(*(columns[0]),","))!=NULL)
                strcpy(data->address,temp);
                    else strcpy(data->address,"");

                     if((temp=strtok(*(columns[1]),","))!=NULL)
            strcpy(data->email,temp);
            else strcpy(data->email,"");

                     if((temp=strtok(*(columns[2]),","))!=NULL)
            strcpy(data->phoneNo,temp);
            else strcpy(data->phoneNo,"");

            *status=1;

      }
      for(i=0;i<noColumns;++i)
      {
            free(*(columns[i]));
      }

          /* Form the query to get the list of services running for the given unique business id */
      sprintf(query,"SELECT M.%s,M.%s FROM %s S, %s M WHERE S.%s ='%s' AND S.%s=
M.%s",WSM_SERVICE_NAME,WSM_SERVICE_SERVICEID,WSR_BUSINESS,WSM_SER
VICE,WSR_BUSINESS_BUSID,Busid,WSR_BUSINESS_SERVICEID,WSM_SERVICE_SER
VICEID);

          if(debug)printf("In getbusidInfo with query *%s*\n",query);

          noColumns=2;
             for(i=0;i<noColumns;++i)
      {
        columns[i]=(char **)malloc(sizeof(char**));

}


     /* call the function to execute the query and return the results */
st = fetch(query,columns,noColumns);

     /* If error in querying the database */
     if(st==-1 || st==0)
     {
             if(st==-1)
                     strcpy(result,"ERROR:Contact Administrator");
             else strcpy(result,"ERROR:No services found for the given Busid");

             serviceName->first=0;
     serviceName->size=strlen(result)+1;
     strcpy(serviceName->array,result);
              serviceId->first=0;
        serviceId->size=strlen(result)+1;
        strcpy(serviceId->array,result);
             *status=1;

             for(i=0;i<noColumns;++i)
          {
          free(*(columns[i]));
     }
             return;
     }
     else    /* Else copy the names and ids to be sent to the client */
     {

             serviceName->first=0;
        serviceName->size=strlen(*(columns[0]))+1;
        strcpy(serviceName->array,*(columns[0]));
         serviceId->first=0;
        serviceId->size=strlen(*(columns[1]))+1;
        strcpy(serviceId->array,*(columns[1]));
             *status=0;

     }
for(i=0;i<noColumns;++i)
{
        free(*(columns[i]));
}

     if(debug)printf("Names *%s* IDs *%s*\n",serviceName->array,serviceId->array);
     /*sprintf(msg,"GetBusidInfo exited for busid %s with status %d\n",Busid,*status);
     tmsg(msg);*/

return;
```

```
}

/*************************************************************************
 * Remote getServiceInfo function in the DCEVEC Server Interface
 *
 * Purpose:  To get information for a given web-service .
 * Function: To prepare queries to query the database and retrieve information for a given web-
service.It queries the database to
 *             retrieve the service Urls, standby Urls, etc.
 * Returns:  return string of the service replicas found.
 *          return string of the respective creation timestamps found.
 *              return string of the standby Urls.
 *         return code to indicate the status.
 ************************************************************************/
void getServiceInfo(
            handle_t  h,       /* DCE communication handle */
       idl_char *ServiceName, /* Name of the Service*/
           idl_char *Busid,           /* Unique business id for the client */
          conarray *replica,          /* Comma separated replica paths to be sent to the client */
           conarray *replicatime, /* Respective comman separated creation timestamps to be sent to
the client */
           conarray *standby,        /* Comma separated Standby paths to be sent to the client */
           /*conarray *failure,
          conarray *failuretime,*/
           unsigned32 *status        /* Status of execution */
)
{
          char query[512]; /* SQL query to query the database */
      char result[64];
      int st,i;
      char **columns[5] /* columns to be populated by the query */
          ,*temp;
      int noColumns;       /* Number to columns to be populated by the query */
          char msg[256];
          char serviceid[64];


          memset(serviceid,'\0',64);
          /*sprintf(msg,"getServiceInfo called by user with busid %s and service name
%s\n",Busid,ServiceName);
      tmsg(msg);*/

          noColumns=1;
                  sprintf(query,"select B.%s from %s A, %s B, %s C where A.%s = '%s' AND
B.%s = '%s' AND B.%s =
C.%s",WSM_SERVICE_SERVICEID,WSR_AUTH,WSM_SERVICE,WSR_BUSINESS,WSR_
AUTH_BUSID,Busid,WSM_SERVICE_NAME,ServiceName,WSM_SERVICE_SERVICEID,
WSR_BUSINESS_SERVICEID);
```

```
        if(debug)printf("In getServiceInfo with query *%s*\n",query);

    for(i=0;i<noColumns;++i)
    {
        columns[i]=(char **)malloc(sizeof(char**));

    }

        /*check if the service exists */
    st = fetch(query,columns,noColumns);


        /* If service doesn't exist, exit */
        if(st==-1 || st==0)
        {
                if(st==-1)
               strcpy(result,"ERROR:Contact Administrator");
          else strcpy(result,"ERROR:Service not found for the given service name");

          replica->first=0;
          replica->size=strlen(result)+1;
          strcpy(replica->array,result);
                *status=1;

                replicatime->first=0;
                replicatime->size=0;
                standby->first=0;
                standby->size=0;
                /*failure->first=0;
                failure->size=0;
                failuretime->first=0;
                failuretime->size=0;*/

                if(debug)printf("Status %d result *%s*\n",*status,replica->array);
        /*sprintf(msg,"GetServiceInfo exited for busid %s with status %d and string
%s\n",Busid,*status,replica->array);
                tmsg(msg);*/

                for(i=0;i<noColumns;++i)
        {
                        free(*(columns[i]));
        }

                return;

        }
        else
        {
                strcpy(serviceid,strtok(*(columns[0]),","));
        }
        for(i=0;i<noColumns;++i)
          {
               free(*(columns[i]));
          }
        /* get the number of servers running for a given webservice */

        noColumns=2;
     sprintf(query,"SELECT S.%s,TO_CHAR(S.%s,'DD-MON-YYYY-HH24:MI') FROM %s S,
%s M WHERE S.%s= M.%s AND
S.%s='%s'",WSM_REPLICA_SERVICEURL,WSM_REPLICA_TIMESTAMP,WSM_REPLIC
A,WSM_SERVICE,WSM_REPLICA_SERVICEID,WSM_SERVICE_SERVICEID,WSM_REP
LICA_SERVICEID,serviceid);

        if(debug)printf("fetch with query *%s*\n",query);

         for(i=0;i<noColumns;++i)
     {
         columns[i]=(char **)malloc(sizeof(char**));

     }

     st = fetch(query,columns,noColumns);


     if(st==-1) /* if error in querying the database */
     {
               strcpy(result,"ERROR:Contact Administrator");
                         replica->first=0;
                 replica->size=strlen(result)+1;
                 strcpy(replica->array,result);
                 *status=1;

         replicatime->first=0;
         replicatime->size=0;
         standby->first=0;
         standby->size=0;
         /*failure->first=0;
         failure->size=0;
         failuretime->first=0;
         failuretime->size=0;*/

                 for(i=0;i<noColumns;++i)
        {
                 free(*(columns[i]));
        }

                  if(debug)printf("Status %d result *%s*\n",*status,replica->array);
         /*sprintf(msg,"GetServiceInfo exited for busid %s with status %d and string
%s\n",Busid,*status,replica->array);
         tmsg(msg);*/
                 return;
```

```
	}
	else
	{
		    if(st>0) /* if atleast one copy of the web-service is running */
		    {
			replica->first=0;
		replica->size=strlen(*(columns[0]))+1;
		strcpy(replica->array,*(columns[0]));
			replicatime->first=0;
		replicatime->size=strlen(*(columns[1]))+1;
		strcpy(replicatime->array,*(columns[1]));
		    }
		    else
		    {
			strcpy(result,"ERROR:No web-servies running for the given service 
name");
			replica->first=0;
		replica->size=strlen(result)+1;
		strcpy(replica->array,result);
		replicatime->first=0;
		replicatime->size=strlen(result)+1;
		strcpy(replica->array,result);
		    }

	    }
	for(i=0;i<noColumns;++i)
	{
	     free(*(columns[i]));
	}

/* get the standby servers for the service */
	   noColumns=1;
	sprintf(query,"SELECT S.%s FROM %s S, %s M WHERE S.%s= M.%s AND 
S.%s='%s'",WSM_STANDBY_GENERICURL,WSM_STANDBY,WSM_SERVICE,WSM_ST
ANDBY_SERVICEID,WSM_SERVICE_SERVICEID,WSM_STANDBY_SERVICEID,servicei
d);

	  for(i=0;i<noColumns;++i)
	{
	     columns[i]=(char **)malloc(sizeof(char**));

	}
	 if(debug)printf("fetch with query *%s*\n",query);

	st = fetch(query,columns,noColumns);


	if(st==-1) /* if error in querying the database */
	{
		strcpy(result,"ERROR:Contact Administrator");
		replica->first=0;
```

```
                replica->size=strlen(result)+1;
                strcpy(replica->array,result);
                *status=1;

            replicatime->first=0;
            replicatime->size=0;
            standby->first=0;
            standby->size=0;
            /*failure->first=0;
            failure->size=0;
            failuretime->first=0;
            failuretime->size=0;*/

            for(i=0;i<noColumns;++i)
            {
                free(*(columns[i]));
            }

                 if(debug)printf("Status %d result *%s*\n",*status,replica->array);
            /*sprintf(msg,"GetServiceInfo exited for busid %s with status %d and string
%s\n",Busid,*status,replica->array);
            tmsg(msg);*/
            return;
        }
        else
        {
            if(st>0) /* If atleast one path found */
            {
                            standby->first=0;
                standby->size=strlen(*(columns[0]))+1;
                strcpy(standby->array,*(columns[0]));
            }
            else
            {
                strcpy(result,"ERROR:No standby servers for the given service name");
                  standby->first=0;
                standby->size=strlen(result)+1;
                strcpy(standby->array,result);
                  }

        }
        for(i=0;i<noColumns;++i)
        {
            free(*(columns[i]));
        }

          *status=0;

          if(debug)printf("Status %d replica *%s* replica time *%s* standby
*%s*\n",*status,replica->array,replicatime->array,standby->array);
        /*      sprintf(msg,"GetServiceInfo exited for busid %s with status %d\n",Busid,*status);
```

```
            tmsg(msg);*/
          return;
}

/***********************************************************************
 * Remote getAccounts function in the DCEVEC Server Interface
 *
 * Purpose:  To get a list of all the accounts.
 * Function: To prepare a query to query the database for the all the services running for a given
unique business id.
 * Returns:  return string of the service names found.
 *            return string of the respective service ids found.
 *            return code to indicate the status.
***********************************************************************/

void getAccounts(
          handle_t  h,       /* DCE communication handle */
            conarray *userName, /* Comma separated service names to be sent to the client */
          conarray *userId,      /* Respective Comma separated service ids to be send to the client
*/
           unsigned32 *type,       /* type of account, i.e. WSM or WSR */
                    unsigned32 *status        /* return code */
)
{

      char query[512];
      char result[64];
      int st,i;
      char **columns[2], /* Columns to be populated by the query */
      *temp;
      int noColumns;   /* Number of Columns to be populated by the query */
      char msg[256];


          /* Form the query to get the account info */
      if(*type==WSR)
                    sprintf(query,"SELECT %s,%s FROM
%s",WSR_AUTH_NAME,WSR_AUTH_BUSID,WSR_AUTH);
          else
                    sprintf(query,"SELECT %s,%s FROM
%s",WSM_AUTH_NAME,WSM_AUTH_BUSID,WSM_AUTH);

           if(debug)printf("In getAccounts with query *%s*\n",query);

      noColumns=2;
         for(i=0;i<noColumns;++i)
      {
            columns[i]=(char **)malloc(sizeof(char**));

      }
```

```
/* call the function to execute the query and return the results */
st = fetch(query,columns,noColumns);

/* If error in querying the database */
if(st==-1)
    {
            *status=1;
            strcpy(result,"ERROR:Contact Administrator\n");
            userName->first=0;
      userName->size=strlen(result)+1;
      strcpy(userName->array,result);
       userId->first=0;
      userId->size=strlen(result)+1;
      strcpy(userId->array,result);

    }
    else if (st==0)
    {
            *status=1;
            strcpy(result,"No Accounts found\n");
      userName->first=0;
      userName->size=strlen(result)+1;
      strcpy(userName->array,result);
       userId->first=0;
      userId->size=strlen(result)+1;
      strcpy(userId->array,result);
    }
    else
    {
            if(debug)
      {
           printf("*%s* *%s*\n",*(columns[0]),*(columns[1]));
      }

            userName->first=0;
      userName->size=strlen(*(columns[0]))+1;
      strcpy(userName->array,*(columns[0]));
       userId->first=0;
      userId->size=strlen(*(columns[1]))+1;
      strcpy(userId->array,*(columns[1]));
      *status=0;

}


     for(i=0;i<noColumns;++i)
      {
           free(*(columns[i]));
      }
```

```
          if(debug)printf("Names *%s* IDs *%s*\n",userName->array,userId->array);
       /*sprintf(msg,"getAccounts exited with status %d\n",*status);
       tmsg(msg);*/

          return;

}
/**********************************************************************
 * Remote authen function in the DCEVEC Server Interface
 *
 * Purpose:  To authenticate a given client .
 * Function: Calls a function that authenticates the client.
 * Returns:  return string.
 *          return code to indicate the status.
 **********************************************************************/

void authen(
  /*   [in]  */       handle_t h,
       accountInfo *data,
  /*  [out,string]*/      idl_char *rstatus,
                           unsigned32 *status
)
{
          char result[64];
          char msg[256];

          /*sprintf("Authen called for user %s\n",data->name);
          tmsg(msg);*/

          /* call the function passing it the username and password */
          *status = authenticateAdmin(data->name,data->word,result);

          /* get the generic Paths */

          strcpy(rstatus,result);

          /*sprintf(msg,"User %s Authenticated- status  %d\n",data->name,*status);
       tmsg(msg);*/

          if(debug)printf("User %s Authenticated- status  %d\n",data->name,*status);
          return;
}

/**********************************************************************
 * Remote getGenericServers function in the DCEVEC Server Interface
 *
 * Purpose:  To authenticate a given client .
 * Function: Calls a function that authenticates the client.
 * Returns:  return string.
 *          return code to indicate the status.
 **********************************************************************/
```

```
void getGenericServers(
  /*   [in]  */       handle_t h,
  /*  [in,string]*/      idl_char *busid,
    conarray *genpaths,       /* Comma separated genericPaths to be sent to the client */
                    unsigned32 *status
)
{
        char result1[64];
        char msg[256];
        char query[256];
        char **columns[1]; /* columns to be populated by the query */
        int noColumns,i,st;   /* Number to columns to be populated by the query */

        /*sprintf("getGenericServers called for busid %s\n",busid);
        tmsg(msg);*/

        /* get the generic Paths */
                noColumns=1;
                sprintf(query,"SELECT %s FROM %s where
%s='%s'",WSM_GENERIC_GENERICURL,WSM_GENERIC,WSM_GENERIC_BUSID,busid
);

                  for(i=0;i<noColumns;++i)
                {
                        columns[i]=(char **)malloc(sizeof(char**));
                           }
                if(debug)printf("In getGenericServers fetch with query *%s*\n",query);

                st = fetch(query,columns,noColumns);

                if(st==-1)
                {
                        strcpy(result1,"ERROR:Contact Administrator");
                        genpaths->first=0;
                        genpaths->size=strlen(result1)+1;
                        strcpy(genpaths->array,result1);
                          *status=1;
                  }
                else
                {
                        if(st>0) /* If atleast one path found */
                        {
                                genpaths->first=0;
                                genpaths->size=strlen(*(columns[0]))+1;
                                strcpy(genpaths->array,*(columns[0]));
                          *status=0;
                                      }
                        else
                        {
                                strcpy(result1,"ERROR:No standby servers for the given service name");
                    genpaths->first=0;
                    genpaths->size=strlen(result1)+1;
                    strcpy(genpaths->array,result1);
                  *status=1;
                 }

             }

             for(i=0;i<noColumns;++i)
             {
                 free(*(columns[i]));
                  }


     /*sprintf(msg,"getGenericServers exited with status %d for busid %s",*status,busid);
     tmsg(msg);*/

     if(debug)printf("getGenericServers exited with status %d for busid %s\n",*status,busid);
     return;
}

/*****************************************************************************
 * Remote startStandbyServer function in the DCEVEC Server Interface
 *
 * Purpose:  To start a standby server for a given service.
 * Function: Forks a child and the child calls a function that starts a standby server for the service.
 * Returns:  return string.
 *****************************************************************************/

void startStandbyServer(
 /*   [in]   */         handle_t h,
 /*   [in,string,ptr] */  idl_char *wsrid,
 /*   [in,string,ptr] */  idl_char *replicapath,
 /*   [out,string]    */  idl_char rstatus[64]
)

{
        int ret,pid,status;
        char result[64];
        char msg[256];

        if(debug)printf("In startStandbyStart with service id *%s* and path
*%s*\n",wsrid,replicapath);

        /*sprintf(msg,"startStandbySErver called for service id %s\n",wsrid);
        tmsg(msg);*/

        if((pid=fork())==0)
     {
                ret = standbyStart(wsrid,replicapath,result);
                exit(ret);
```

```
            }
            waitpid(pid,&status,0);
            ret=status>>8;
            /*sprintf(msg,"startStandbySErver exited with status %d for service id %s\n",ret,wsrid);
        tmsg(msg);*/

            strcpy(rstatus,"OK");
}

/**********************************************************************
            ****SECURITY CODING****
 This function is imported from the example file provided by the instructor.
 * DCEVEC_STOP_SERVER - Stop the server from further request handling
 *********************************************************************/
void wsm_stop_server(
  /*  [in]           */ handle_t     h,          /* DCE RPC Handle */
  /*  [in,string,ptr] */ idl_char     *lname,     /* Userid of requestor */
  /*  [in,string,ptr] */ idl_char     *validity,  /* Validity check value */
  /*  [out]          */ unsigned32   *stat       /* Status code on return */
  )
{           char  amsg[144];         /* message build area */
            char  *pvalue="cs481.";  /* validity check string */
/* Say who made request */

  sprintf(amsg,
    "wsm_stop_server: Request to stop server\n\tuser=%s validity=%s",
    lname, validity);
  p_handle(h);

/* Check for a valid request */
  if (strcmp((const char *)pvalue, (const char *)validity)) {
    /* user specified invalid keyvalue - don't let them through */
    tmsg("wsm_stop_server: Rejected stop request\n");
    *stat = 2;  /* set invalid request status */
    return;  /* and return to the requestor */
    }

/*
 * User seems to be ok ...
 * Start the shutdown and cleanup processes.
 */
  rpc_mgmt_stop_server_listening(NULL, stat);

/* Record the shutdown attempt in the server's log */
  sprintf(amsg, "wsm_stop_server: Stopping Server - ST=%08x\n\t%s\n",
    *stat, dcemsg(*stat));
            tmsg(amsg);

/* Return to caller with the outcome of the request in "stat" */
  return;
```

```
}

void handler(int x)
{
	printf("Alarm went off1\n");
/*	signal(SIGALRM, handler);*/
}
```

Multicast Listener

1) Mcast.h

```
#include <sys/types.h>
#include <pthread.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
/* remove 8/28/00
#ifdef SOLARIS
#include <sys/sockio.h>
#else
#include <linux/sockios.h>
#endif
*/

/* #include <sys/ioctl.h> */

#define DefaultPort   5050
#define DefaultGroup "224.1.1.100"
#define DefaultTTL   64
#define DefaultSamples 50
#define MaxMsg 2048
#define DEFAULTINTERFACE ""
#define DefaultOraclePort 1521
#define DefaultOracleSID "darwin"
#define DefaultHost "darwin0"

#define DEBUG 1

#define ALLOW "128.206"

/* Message structure */
typedef struct {
  unsigned int  type;
  unsigned int  App;
  unsigned int  msglen;
#define AppHdrLen 12
/*              type     + msglen */
  char msg[MaxMsg];
} Message_t;

typedef struct {
  int sfd, rfd;
  int nsamples;
  struct sockaddr_in maddr;
  struct sockaddr_in iface;
  char myhost[128];
  char mgroup[32];
} Control_t;
```

2) Listener.c

```
/**********************************************************************
 *Program listener:  listener.c
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  To listen on a given multicast address.
 *
 * Function: It listens for announcements on a given multicast address. Takes the necessary action
on receiving
 *              a message and sends a result string back to the sender of the multicast message.
 *
 **********************************************************************/

#include "mcast.h"
#include "database.h"
Control_t Control;
void handler();
void handler1();
int executeSQL(Message_t ,char *,char *);
void checkdatabase(char*,char*);
int sendMulticast(Message_t *msend, char *result);

char db_String[256];

int debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

struct datatothread{
char dbString[256];
Message_t message;
struct sockaddr_in addr;
};

void messageDecode(char *string)
```

```
{
       int i;
       for(i=0;i<strlen(string);++i)
       string[i]=string[i]-128;
}


/***********************************************************************
 * SendWSRAddress
 *
 * Purpose:  To send the latest WSR web interface addresses to the sender of the multicast
message.
 * Function: It is called on the creation of a thread. It is passed the address of the sender.
 *             It queries the database to get the latest access points of WSR web interface and sends it
to the specified address.
 ***********************************************************************/

void * SendWSRAddress(void *x)
{

  struct datatothread data;

  char result[128];
  int datalen, msglen, addrlen, n;
  int noColumns,i,st;
  char **columns[1];
  char        query[256];
  char    msend[1024];

           memset(msend,'\0',1024);
       data = *(struct datatothread *)x;


   sprintf(query,"SELECT %s FROM %s  where %s
='%s'",WSR_ACCESSPOINTS_URL,WSR_ACCESSPOINTS,WSR_ACCESSPOINTS_SERVI
CEID,CGISERVER_SERVICEID);

           if(debug)printf("Query %s\n",query);

  noColumns=1;
  for(i=0;i<noColumns;++i)
       {
            columns[i]=(char **)malloc(sizeof(char**));

       }

       st = fetch(query,columns,noColumns);

       if(st==0 || st==-1)
       {
            strcpy(msend,"ERROR:");
```

```
        }

            if(st>0)
            {
                     strcpy(msend,*(columns[0]));
            }
                     for(i=0;i<noColumns;++i)
              {
                    free(*(columns[i]));
              }


   addrlen = sizeof(struct sockaddr_in);

   /* Now send reply message back to sender */
   if ((n = sendto(Control.rfd,
               (const void *)msend,
               strlen(msend)+1, 0,
               /*(struct sockaddr *)&Control.maddr,*/
                (struct sockaddr *)&data.addr,
               addrlen)) < 0) {
     /* opps - the send failed - get out of here */
     perror("Reply to sender failed!\n");
    pthread_exit(0);
     return 0;
   }
   if(debug)printf("WSR Address(s) *%s* sent back \n",msend);
      pthread_exit(0);
  return 0;
}

/*************************************************************************
 * SendConnectString
 *
 * Purpose:  To send the dbString (String used to remotely connect to the database on this server)
to the sender of the multicast message.
 * Function: It is called on the creation of a thread. It is passed the address of the sender and the
dbString to be sent.
 *                    It sends the message to the specified address.
 *************************************************************************/

void * SendConnectString(void *x)
{

  struct datatothread data;

  char result[128];
  int datalen, msglen, addrlen, n;
  Message_t msend;
```

```
            data = *(struct datatothread *)x;

  memset ((void *)&msend, 0, sizeof(Message_t));
                        strcpy(msend.msg,data.dbString);
                        msend.type=htonl(DBASE_QUERY);
                        msend.App=data.message.App;
            msend.msglen = htonl((msglen = strlen(msend.msg)+1));
    datalen = AppHdrLen + msglen;
    addrlen = sizeof(struct sockaddr_in);

    /* Now send reply message back to sender */
    if ((n = sendto(Control.rfd,
                (const void *)&msend,
                datalen, 0,
                /*(struct sockaddr *)&Control.maddr,*/
                 (struct sockaddr *)&data.addr,
                addrlen)) < 0) {
      /* opps - the send failed - get out of here */
      perror("Reply to sender failed!\n");
     pthread_exit(0);
      return 0;
   }
    if(debug)printf("Connect string *%s*  sent back \n",msend.msg);
            pthread_exit(0);
  return 0;
}

/*********************************************************************
 * QueryExecute
 *
 * Purpose:  To send the result of executing a query to the sender of the multicast message
(query).
 * Function: It is called on the creation of a thread. It is passed the address of the sender and the
query to be executed.
 *             It sends the result to the specified address.
 *********************************************************************/
void * QueryExecute(void *x)
{

  struct datatothread data;

  int datalen, msglen, addrlen, n;
  Message_t msend;

       data = *(struct datatothread *)x;

  memset ((void *)&msend, 0, sizeof(Message_t));
  memset(msend.msg,'\0',MaxMsg);
            executeSQL(data.message,data.dbString,msend.msg);
              msend.type=htonl(UPDATE);
                        msend.App= data.message.App;
      msend.msglen = htonl((msglen = strlen(msend.msg)+1));
    datalen = AppHdrLen + msglen;
    addrlen = sizeof(struct sockaddr_in);

    /*sleep(3);*/
    /* Now send reply message back to sender */
    if ((n = sendto(Control.rfd,
                (const void *)&msend,
                datalen, 0,
                /*(struct sockaddr *)&Control.maddr,*/
                 (struct sockaddr *)&data.addr,
                addrlen)) < 0) {
      /* opps - the send failed - get out of here */
      perror("Reply to sender failed!\n");
     pthread_exit(0);
      return 0;
   }
    if(debug)printf("Result String *%s* sent \n",msend.msg);
       pthread_exit(0);
  return 0;
}

/***********************************************************************
 * WriteTempTransactions
 *
 * Purpose:  To send the result of executing a query to the sender of the multicast message
(query).
 * Function: It is called on the creation of a thread. It is passed the address of the sender and the
query to be executed.
 *            It sends the result to the specified address.
 ***********************************************************************/
void * WriteTempTransactions(void *x)
{

            char result[256];
            int ret;
  struct datatothread data;
            memset(result,'\0',256);

       data = *(struct datatothread *)x;

       ret = writeTempTransactions(data.dbString,data.message,result);

            pthread_exit(0);
  return 0;
}

/***********************************************************************
 * listener
 *
 * Purpose:  To listen for multicast announcements and serve the requests.
 * Function: It listens for messages, parses the messages and creates threads to handle the
requests.It passes the address of the
 *              sender to the threads so they can send the results back to the sender.
 *********************************************************************/

int listener() {
  Message_t mrecv;
  struct sockaddr_in raddr; /* mcast address we are receiving on */
  time_t t;
  int n,addrlen;
  pthread_t tid;
    pthread_attr_t attr;
    struct datatothread data[5];
        int count=0;
             int datalen, msglen;
             char syncflag[256];


             memset(syncflag,'\0',256);

             if(debug)printf("Listener started\n");

    if (pthread_attr_init(&attr) != 0) {
      perror("Error initing pthread attr\n");
      exit(1);
    }

    if (pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM) != 0) {
      perror ("Error initializing pthread attr scope to PTHREAD_SCOPE_SYSTEM\n");
      exit(1);
    }

        if(pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED)!=0) {
         perror ("Error setting PTHREAD_CREATE_DETACHED\n");
      exit(1);
    }

  memset ((void *)&mrecv, 0, sizeof(Message_t));

while (1) {

    addrlen = sizeof(struct sockaddr_in);
    /* attempt to receive a message from MC address */
    if ((n = recvfrom(Control.rfd,
                  (void *)&mrecv,
                  sizeof(Message_t), 0,
                  (struct sockaddr *) &raddr,
                  &addrlen)) < 0) {
      perror("recvfrom");
      return 1;
    }
```

```
                    CheckSyncFlag(syncflag);

             /*Decode the message received*/
           messageDecode(mrecv.msg);
    /* Note the fact we received a message */
    printf("Received msg from %s at time %-24.24s\n\t*%s*%hd\n",
            inet_ntoa(raddr.sin_addr), ctime(&t), mrecv.msg,ntohl(mrecv.type));

/* Call the function to execute the SQL statement */
             if(ntohl(mrecv.type) == UPDATE){
                      if(debug)printf("Update message received\n");

                      /* Check to see if the update message is coming from know hosts. IF from
unknown host, don't entertain the request */
                      if(strncmp(inet_ntoa(raddr.sin_addr),ALLOW,7))
                      {
                               printf("Unknown host %s sending multicast
message\n",inet_ntoa(raddr.sin_addr));
                               continue;
                      }
                      strcpy(data[count].dbString,db_String);
                  data[count].message=mrecv;
                      data[count].addr = raddr;

                      if(!strcmp(syncflag,"false")) /* Synchronisation done */
                      {
                               printf("***Calling QueryExecute\n");
                               if ( pthread_create(&tid, &attr, QueryExecute, &data[count]) != 0 ) {
                      perror ("Error creating first pthread\n");
                      exit(1);
                      }
                      }
                      else if (!strcmp(syncflag,"true")) /* Synchronisation taking place */
                      {
                               printf("***Calling WriteTempTrans\n");
                                if ( pthread_create(&tid, &attr, WriteTempTransactions, &data[count])
!= 0 ) {
                    perror ("Error creating first pthread\n");
                    exit(1);
                    }
                      }
             count++;
             }
/* Call the function to get the information needed by the client to connect to the ORACLE server
*/

             if(!strcmp(syncflag,"false"))
             {
                      if(ntohl(mrecv.type) == DBASE_QUERY){
                               if(debug)printf("Database Query message received\n");
```

```
                                strcpy(data[count].dbString,db_String);
                                data[count].message=mrecv;
                                data[count].addr = raddr;
                                if ( pthread_create(&tid, &attr, SendConnectString, &data[count]) != 0 )
{
                                perror ("Error creating first pthread\n");
                                exit(1);
                        }
                        }
                        if(ntohl(mrecv.type) == WSR_QUERY){
                        if(debug)printf("WSR Query message received\n");
                        strcpy(data[count].dbString,db_String);
                        data[count].message=mrecv;
                        data[count].addr = raddr;
                        if ( pthread_create(&tid, &attr, SendWSRAddress, &data[count]) != 0 ) {
                        perror ("Error creating first pthread\n");
                        exit(1);
                        }

                        }

        count++;
        }
        count=count%5;


  }/* while loop ends*/

}

/*****************************************************************************
Funtion: main
Purpose: To create and initialize a socket to listen to multicast announcements.
******************************************************************************
/
int pid,pid1;

int main (int argc, char *argv[]) {
  const int on = 1;
  const unsigned char loop = 0;
  char c, interface[32];
  int mport = DefaultPort;
  unsigned char ttl = 64;
  struct ip_mreq mreq;
  char OracleHost[32];
  int OraclePort = -1;
  char OracleSID[32];
  char result[128];
  char Method[16];
  int       ret;
  char db_String_remote[256];
  int parentid;

            memset(result,'\0',128);
            memset(Method,'\0',16);

/* Init control structure */
  memset ((void *)&Control, 0, sizeof(Control));
  memset(interface,'\0',32);
  memset(OracleHost,'\0',32);
  memset(OracleSID,'\0',32);
  memset(db_String,'\0',128);

  bzero(&Control,sizeof(Control_t));

            signal(SIGINT,handler);

/* Determine who we are */
  if (gethostname((char *)Control.myhost, 128) == -1) {
    perror ("Error getting local host name!\n");
    exit(1);
  }

  /* Initialize all structs */
  strncpy ((char *)Control.mgroup, DefaultGroup, 31);

            while ((c = getopt(argc,argv, "H:S:P:g:p:i:I:")) != -1) {
                    switch ((char) c) {
                            case 'g':
                            strncpy (Control.mgroup, optarg, 31);
                            break;
                            case 'p':
                                    if ((mport = atoi(optarg)) < 0) {
                            printf("Error getting port number\n");
                            exit(0);
                            }
                            break;
                            case 'i':
                            strncpy (interface, optarg, 31);
                            break;
                            case 'H':
                        strncpy (OracleHost, optarg, 31);
                        break;
                  case 'P':
                        if ((OraclePort = atoi(optarg)) < 0) {
                        printf("Error getting Oracle port number\n");
                        exit(0);
                        }
                        break;
                  case 'S':
                        strncpy (OracleSID, optarg, 31);
                        break;
                    case 'I':
                            strcpy(Method,optarg);
                            break;


            }

        }

  if((strcmp(Method,"T") && strcmp(Method,"C") && strcmp(Method,"N")) ||
strlen(Method)==0)
  {
        printf("Usage: listener -I importMethod [-g] GroupAddress [-p] port number [-i] interface
[-H] OracleHost [-P] OraclePort [-S] OracleSid\n");
        printf("where importMehtod is either T to execute new transactions to sync the database
or C to copy entire tables to sync the database or N for no synchronization and refresh\n");
     printf("Specify either T or C with the -I flag\n");
        exit(1);
  }

  if(strlen(OracleHost)==0)
        strcpy(OracleHost,DefaultHost);
  if(strlen(OracleSID)==0)
        strcpy(OracleSID,DefaultOracleSID);

  if(OraclePort==-1)
        OraclePort=DefaultOraclePort;


  sprintf(db_String,"(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL =
TCP)(HOST = %s)(PORT = %d)))(CONNECT_DATA =(SID =
%s)))",OracleHost,OraclePort,OracleSID);

        printf("Checking database %s\n",db_String);
        checkdatabase(db_String,result);
        printf("Checking database done\n");

        if(strcmp(result,"OK"))
        {
                printf("Database on this server is not working\n");
                exit(1);
        }

        if(debug)printf("Found database on this server\n");
 /* First open sockets */
 if ((Control.rfd=socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
   perror("Error opening receiving socket");
   exit(1);
 }

 /* Set the sending and rx'ing multicast interface */
 if (strlen(interface)!=0) {
   if (strlen(interface) > 6) { /* must be an ip address */
     printf("\nUsing the interface with address %s\n\n", interface);
     Control.iface.sin_addr.s_addr = inet_addr(interface);
   } else {
       printf("Only valid IP addresses are accepted here!\n");
       exit(1);
   }
 } else {
   Control.iface.sin_addr.s_addr = INADDR_ANY;
 }

 /* And the receiving side */
 Control.maddr.sin_family = AF_INET;
 Control.maddr.sin_port = htons(mport);
 Control.maddr.sin_addr.s_addr = inet_addr(Control.mgroup);
 if (bind (Control.rfd, (struct sockaddr *)&Control.maddr,
           sizeof (struct sockaddr_in)) < 0) {
   perror("bind error on rx socket!\n");
   close(Control.rfd);
   exit(1);
 }

 mreq.imr_multiaddr.s_addr = inet_addr(Control.mgroup);
 mreq.imr_interface.s_addr = Control.iface.sin_addr.s_addr;
 if (setsockopt(Control.rfd, IPPROTO_IP,
                IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq)) < 0) {
   perror("setsockopt IP_ADD_MEMBERSHIP");
   close(Control.rfd);
 exit(1);
 }


if(strcmp(Method,"N")) /* if N set, then no synchronization and no refresh*/
{
/* Set the flag to true for database synchronisation */
            ret = SetSyncFlag(db_String,result);

             if(strcmp(result,"OK"))
        {
               printf("Database synchronization failed\n");
               exit(1);
        }

 /* Fork a child process to execute database synchronisation */

             if((pid=fork())==0)
              {

                     Message_t msend;
                     int         retcode;
                     char         result[256];

                     memset(result,'\0',256);
                     printf("Synchronisation started\n");

                     /* send Multicast message to get the db_String */
                 memset ((void *)&msend, 0, sizeof(Message_t));

                 memset(msend.msg,'\0',MaxMsg);
            msend.type=ntohl(DBASE_QUERY);

                 retcode = sendMulticast(&msend,result);
                     retcode=0;
                     if ( retcode ==0)
            {
                     strcpy(db_String_remote,result);
                     printf("Found a database to synchronize transactions with having DbString
%s\n",db_String_remote);
            }
                     else
            {
                     printf("Can't find a database to synchronize transactions with\n");
                     printf("Database synchronization failed\n");
                    parentid=getppid();
                    printf("Listener Stopped\n");
                    kill(parentid,15);
                    exit(1);
                     }

                     if(!strcmp(Method,"T"))
                     ret = SyncDatabase(db_String,db_String_remote,result);
                     else if (!strcmp(Method,"C"))
                     ret = CopyDatabase(db_String,db_String_remote,result);

                     if(strcmp(result,"OK"))
                  {
                  printf("Database synchronization failed\n");
                             parentid=getppid();
                             printf("Listener Stopped\n");
                             kill(parentid,15);
                     exit(1);
            }

                     printf("Synchronisation done\n");
                     exit(0);
              }

      /* Fork a child process to execute database Refresh periodically*/

           if((pid1=fork())==0)
            {
                 Message_t msend;
                 int         retcode;
                 char        result[256];

                     signal(SIGINT,handler1);

                     while(1)
                     {
                          sleep(1800); /* sleep for 30 minites */
                     memset(result,'\0',256);
                     printf("Refresh started\n");
                     /* send Multicast message to get the db_String */
                     memset ((void *)&msend, 0, sizeof(Message_t));

                       memset(msend.msg,'\0',MaxMsg);
                  msend.type=ntohl(DBASE_QUERY);

                     retcode = sendMulticast(&msend,result);
                  retcode=0;
                     if ( retcode ==0)
                     {
                     strcpy(db_String_remote,result);
                     printf("Found a database to synchronize transactions with having DbString
%s\n",db_String_remote);
                     }
                     else
                     {
                     printf("Can't find a database to refresh transactions with\n");
                     printf("Database refresh failed\n");
                     }

                       ret = RefreshDatabase(db_String,db_String_remote,result);

                       if(strcmp(result,"OK"))
                  {
                          printf("Database Refresh failed\n");
                     }
                             else
                             {
                                     printf("Database Refresh successful\n");
                             }

                       printf("Database Refresh attempted\n");
                     }

                  exit(0);
            }
}

            /* Listen for the database requests */
```

```
   listener();

  exit(0);
}

void handler()
{

            printf("Shutting down the listener\n");
            kill(pid1,SIGINT);
            close(Control.rfd);
            exit(1);
}

void handler1()
{

        printf("Shutting down the Refresh process\n");
        exit(1);
}
```

Fault Manager

1) Monitor.c

```
/*********************************************************************
 * Fault Manager:  monitor.c
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  Fault Manager
 *
 *
 * Syntax: This server program is invoked with the command line:
 *           wsms [&]   (we should invoke program in background)
 *
 * Change Log:
 *
 *********************************************************************/

#include <mcast.h>
#include <database.h>

int sendMulticast(Message_t *msend, char *result);
int callGenericService(char *method,char *WSRid, char *pathGeneric,char *result);
int checkService(char *serviceid);

char db_String[256];
int debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main()
{

      char **columns[2];
         char **standby[1];
         int noColumnsStandby;
      int noColumns,i,st,retcode;
         Message_t msend;
         char query[256];
         char *wsrid,*temp;
         char result[256];
         char msg[256];
         int *pid,*status;
```

```
    debug=1;
     memset ((void *)&msend, 0, sizeof(Message_t));

memset(msend.msg,'\0',MaxMsg);
memset(result,'\0',256);
memset(db_String,'\0',256);

    msend.type=htonl(DBASE_QUERY);

    retcode = sendMulticast(&msend,result);
    if ( retcode ==0)
{
      strcpy(db_String,result);
      sprintf(msg,"Found a database to connect with having DbString %s\n",db_String);
      tmsg(msg);
}
    else
{
      sprintf(msg,"Can't find a database to connect with\n");
      tmsg(msg);
      exit(1);  /* write error message and unregister everything */
}

    sprintf(query,"select %s From %s",WSM_SERVICE_SERVICEID,WSM_SERVICE);

noColumns=1;

for(i=0;i<noColumns;++i)
{
      columns[i]=(char **)malloc(sizeof(char**));

}


    if(debug)printf("Getting service ids using query *%s*\n",query);

st = fetch(query,columns,noColumns);

    pid=(int*)malloc(sizeof(int)*st);
status=(int*)malloc(sizeof(int)*st);

              for(i=0;i<st;++i)
              {
                      if((temp = strrchr(*(columns[0]),','))!=NULL)
             {
                  *temp='\0';
                  if((temp=strrchr(*(columns[0]),','))!=NULL)
                  {
                        wsrid=temp+1;
                  }
                  else
```

```
                        {
                        wsrid=*(columns[0]);
                        }
                    }
                            if((pid[i]=fork())==0)
                    {
                                    if(debug)printf("Calling child process for wsrid *%s*\n",wsrid);
                                    checkService(wsrid);
                                    if(debug)printf("Out of child for wsrid *%s*\n",wsrid);
                            exit(1);
                            }

                    }

                    if(debug)printf("Waiting for the child processes to finish\n");
                    for(i=0;i<st;++i)
                    {
                            waitpid(pid[i],&status[i],0);
                    }


                    for(i=0;i<noColumns;++i)
            {
                free(*(columns[i]));
            }

            free(pid);
            free(status);
            if(debug)printf("Monitor exiting\n");

}


int checkService(char *serviceid)
{



            char **columns[1];
        int noColumns,i,st,retcode;
        char query[256];
        char *pathService,*temp1,*temp;
        char result[256];



            sprintf(query,"select %s From %s where
%s='%s'",WSM_REPLICA_SERVICEURL,WSM_REPLICA,WSM_REPLICA_SERVICEID,se
rviceid);

        noColumns=1;

        for(i=0;i<noColumns;++i)
        {
               columns[i]=(char **)malloc(sizeof(char**));

        }
        if(debug)printf("In checkservice with wsrid *%s*Calling fetch with query
*%s*\n",serviceid,query);

        st = fetch(query,columns,noColumns);

        if(st !=0 && st!=-1)
        {
                while((temp = strrchr(*(columns[0]),','))!=NULL)
                {
                       *temp='\0';
                       if((temp=strrchr(*(columns[0]),','))!=NULL)
                       {
                              pathService=temp+1;
                       }
                       else
                       {
                             pathService = *(columns[0]);
                       }

                       callGenericService("status",serviceid,pathService,result);

                       if(strncmp(result,"OK",2))
                       {
                              standbyStart(serviceid,pathService,result);
                       }
                }
        }

              for(i=0;i<noColumns;++i)
             {
                 free(*(columns[i]));
             }


}
```

GenericSoapServer

1) GenericSoap.java

```
/*********************************************************************
  GenericSoapServer's function file: genericSoap.java
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  Implements the functions of genericSoapServer
 *
 *
 * Change Log:
 *
 *********************************************************************/

import java.lang.*;
import java.io.*;


public class genericSoap
{

        native int start(String service, String WSRid);
        native String getURL(String WSRid,int processid);
        native int stop(String WSRid);
        native int status(String WSRid);
        public genericSoap()
        {
                /* initialization code */
        }


        static
      {
           String library =
"/usr/users/aakwv5/webservices/testexamples/genericSoapServer/libgenericSoap.so"; // in current
directory
           try
           {
                    System.load( library );
           }
           catch( Exception e )
           {
                    System.out.println( "\nERROR: Unable to load " + library + "\nException: "
+ e + "\n" );
           }
        } // end static


        public String startService(String service, String WSRid)
        {
                int processid;
                String URL;
                genericSoap g = new genericSoap();
                processid = g.start(service,WSRid);

                if(processid ==-1)
                {
                        return "ERROR:Service already running";
                }
                else if(processid == -2)
                {
                        return "ERROR:Service by this name does not exist on the server";
                }

                URL = g.getURL(WSRid,processid);

                if(URL.compareTo("ERROR")==0)
                return "ERROR:Service not started";
                else return URL;
        }

         public String stopService(String WSRid)
      {
        int status;
              genericSoap g = new genericSoap();
          status = g.stop(WSRid);
                if(status==0)
                return "SERVICE STOPPED";
                else return "ERROR:Service not running";

        }

        public String getStatus(String WSRid)
        {
                int procstatus;
                String ret;
          genericSoap g = new genericSoap();
          procstatus = g.status(WSRid);
                if(procstatus==0)
                ret="OK";
                else
                        ret="ERROR:Service not running";
                System.out.println(WSRid+" "+ret);
                return ret;
        }
}
```

GenericSoapClient

1) GenericSoapClient.java

```
/*********************************************************************
  GenericSoapClient: genericSoapClient.java
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose: Calls the functions of genericSoapServer
 *
 *
 * Change Log:
 *
 *********************************************************************/

import electric.registry.Registry;

class CallService extends Thread{

        String Url;
        String func;
        String WSR;
        String result;

        CallService(String Url,String func,String WSR) {
                System.out.println("Constructor");
            this.Url = Url;
        this.func = func;
        this.WSR = WSR;
           result = new String("ERROR");
        }

     public void run() {

                System.out.println("Binding to the .NET Web Service");
                IgenericSoap gen;
                Thread b = Thread.currentThread();
                try
                {
                gen = genericSoapHelper.bind(Url);
                }
                catch(Exception e)
                {
                        result=new String("ERROR:Cannot bind to "+Url);
                        return;
                }
```

```
          if(func.compareTo("start")==0)
      {
         int a = WSR.indexOf(',');
         String name = WSR.substring(a+1);
         String WSRid = WSR.substring(0,a);

            try{
                  System.out.println("beforesleep");
                 result = gen.startService(name,WSRid);
                  System.out.println("AFtersleep");
                   }
           catch(Exception e)
          {
                 result=new String("ERROR:Cannot call service "+Url);
                     System.out.println(e);
          }

      }
      if(func.compareTo("stop")==0)
          {
                 try{
                        result  = gen.stopService(WSR);
            }
           catch(Exception e)
           {
                 result=new String("ERROR:Cannot call service "+Url);
          System.out.println(e);
            }


          }
          if(func.compareTo("status")==0)
      {
                  try{
                        result = gen.getStatus(WSR);
                    }
           catch(Exception e)
           {
                 result=new String("ERROR:Cannot call service "+Url);
          System.out.println(e);
           }
          }

     }
  }


public class genericSoapClient
  {
  public static String main( String[] args )
    throws Exception
  {
        int i,j;
 // bind to .NET web service

        String result = new String("ERROR");
        Thread t = Thread.currentThread();
        int count,timeout=0;

                String genericUrl = new String(args[2]);
                CallService c = new CallService(args[2],args[1],args[0]);
                try
                {
                c.start();
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
                count =0;
                while(c.isAlive()==true && count<7)
                {
                        t.sleep(6000);
                        count++;
                }
                try
                {
                        if(c.isAlive()==true)
                        {
                        /*c.destroy();*/
                        timeout=1;
                        result = new String("ERROR:TIMEDOUT calling "+genericUrl);
                        }
                }
                catch(Exception e)
                {

                }
                if(timeout==0)
                result = c.result;
                return result;
  }
}
```

WsmSoapServer

1) WsmSoap.java

```
/***********************************************************************
  WsmSoapServer's function file: WsmSoap.java
 *
 * Author:   Abdulkadar Khambati
 * Date:     June 2004
 * Place:    CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  Implements the functions of WsmSoapServer
 *
 *
 * Change Log:
 *
 ***********************************************************************/

import java.io.*;
import java.util.*;
import java.lang.*;
import electric.registry.Registry;
import electric.util.Base64;


public class wsmSoap
{

        native String notifyWSM(String wsrid,String replicapath);

        public wsmSoap()
        {

        }


        public void notify(String wsrid,String replicapath)
        {
                System.out.println("Service "+replicapath+" failed");
                String result = notifyWSM(wsrid,replicapath);
        }
        static
      {
             String library =
"/usr/users/aakwv5/webservices/testexamples/wsmSoapServer/libwsmSoap.so"; // in current
directory
             try
             {
                     System.load( library );
             }
		catch( Exception e )
		{
			System.out.println( "\nERROR: Unable to load " + library + "\nException: "
+ e + "\n" );
		}

	} // end static

}
```

Wsmclient

1) Wsmclient.c

```
#pragma option showinc source
/*************************************************************************
 * DCE Program wsmclient:  wsmclient.c - Client Application Program
 *
 * Author:   Khambati Abdulkadar
 * Date:     June 2004
 * Location: CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  DCE client for Web Service Manager
 *
 * Function: It calls the procedures of the Web Service Manager
 *
 *
 * Syntax:   This program is invoked with the command line:
 *              wsmclient  <server_host>
 *          where:
 *           server_host - identifies the host where server is running
 * Example:  wsmclient darwin
 *************************************************************************/
#include <wsmclient.h>

int findserver();    /* Utility function to look up the name service and get the bindings for the
registered servers */
int locserver();    /*  Utility function to Locate an active server on a designated host machine.*/
void client_alloc(), in_pull(),out_push();  /* from stub code */
int getUserInfo(accountInfo *,int);
int getServiceDet(char *service,serviceInfo *serviceDet,char **genpaths,char **standby,int *);
void printTwoCol(char*,char*,char*,char*);
int getUpdateUserInfo(accountInfo *userInfo);
int getUpdateServiceInfo(serviceInfo *serviceDet);
void code(char *);

char tmpstr[256];
char *rdptr;
char backupGeneric[1024];
#define getUinput(inp, size) {  memset((inp), 0, (size)); \
  rdptr = fgets(tmpstr, 256, stdin); \
  if (rdptr != NULL) { if (strlen(rdptr) > (size)) *(rdptr+(size)-1)='\0'; \
      strcpy((inp),rdptr); \
      if(strrchr((inp),'\n')) \
            inp[strlen(inp)-1]='\0'; \
    } \
  else strcpy((inp), "NULL"); \
  }

int debug;
```

```
#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif


main(argc, argv)
  int argc;
  char *argv[];
{
  int num; /* to hold the return value from locserver function */
      unsigned32 st;  /* To hold DCE status code from locserver */
          unsigned32 status;
          int num_entry;
          int           nsrv;  /* number of servers in demo run */
          int           index[MaxSrv+1];  /* Index to servers */
          int           proto[MaxSrv+1]; /* Index to handles we use */
          int              docreate;
          unsigned32 return_val;   /* To hold the return values from the RPC functions */
      unsigned32 type;
          char      *host,         /* hostname for server */
          *objstr;        /*object string value */
          rpc_binding_handle_t rh;           /* To hold the binding handle to communicate with the
server */
          rpc_binding_vector_t *results;
          char         group_name[NMLEN];
          pipe_state state;         /* state of the pipe */
          pipe_c     data;          /* a pipe structure is allocated */
          char       lcl_source[100];  /* to hold the name of the file to be sent through pipe  */
          char       lcl_target[100];  /* to hold the name of the target file for file received through
pipe*/
          char         namestr[32];
          idl_char               *serviceName;
          char         service[32];
          file_spec  cbind_h;       /* customized binding handle */
          int        i,j;
          unsigned32 rc=0;
          char       **genpaths,*temp,*temp1,*value,*value1;
          idl_char   *paths;
          char       **standby;
      idl_char   *standbypaths;
          char         *namptr;
          conarray
resultback,serviceNames,serviceIds,replicas,replicatimes,standbyservers,genericServers;
          idl_char   busid[64];
          accountInfo userInfo;
          serviceInfo serviceDet;
          char      menu[8];
          int        flagAuthen=0,serviceCount;
          char    statusString[256];
          char      dummy[32];
          clock_t   clock(),startcputime,endcputime;
       time_t       startexectime,endexectime;

          time(&startexectime);
          startcputime = clock();
          host = strdup(argv[1]); /* move Server hostname */
          objstr = strdup(OBJSTR); /* copy the Server's DCE object UUID */
          strcpy(group_name, GRPNAME);  /* identify the group desired */

          memset(namestr,'\0',24);
          memset(service,'\0',32);


#ifndef NSREG
          num = locserver(host, objstr, &rh, &st);  /* call the locserver to locate server on host */

          if (num == 0) {                      /* if unable to locate server on host print message and exit
*/
       printf("Failed to communicate with server on %s\n\t%s\n",
        host, dcemsg(st));
       exit(1);
       }
          time(&endexectime);
          endcputime=clock();
if(debug)printf("Time elapsed to find DCE servers to communicate with cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);

#else
/*********************************Look up in the name
service***************************************************/
          num_entry = findserver(group_name, objstr, C_IFSPEC, &results, MaxSrv,
            &st);  /* All work is done in this routine */

          if (num_entry == 0)  {  /* No servers found registered */
     printf("Lookup for WSM Servers Failed - %s\n", dcemsg(st));
     }

/*
 * Sort out the unique servers in the list of handles
 */
  j = s_handle(num_entry, &nsrv, results, index, proto);

/*
 *  For each server found running, complete the binding information
 */
  for (i=0; i<nsrv; i++)  {
     rpc_mgmt_set_com_timeout(results->binding_h[proto[i]],
        rpc_c_binding_min_timeout, &st); /* timeout quickly */
     rpc_ep_resolve_binding(results->binding_h[proto[i]],
                        C_IFSPEC, &st);
      }  /* End for each server loop */
time(&endexectime);
endcputime=clock();
if(debug)printf("Time elapsed to find DCE servers to communicate with cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);


          for (i=0; i < nsrv; i++) { /* Output the menu for servers */
          printf("%2d.",i);
       p_handle(results->binding_h[proto[i]]); /* print who we talk to */
       }

          printf("\n\nPick the server to communicate with. Enter the number\n");
          memset(menu,'\0',sizeof(menu));
       getUinput(menu,sizeof(menu));
          rh = results->binding_h[proto[atoi(menu)]];
          printf("Communicating with the web service manager on...\n");
          p_handle(rh);

/*****************************************************************************
*******************************************/
#endif

  flagAuthen=1;
  memset(busid,'\0',64);
   do
          {
                  do
                  {
                          system("clear");
                  printf("Enter one of the following\n");
                          printf("1. Menu to create or delete a webservice cluster OR start or stop a
webservice instance\n");
                          printf("2. Update the list of back-up servers for a webservice cluster\n");
                          printf("3. Get Info on the Account and the web service cluster(s)\n");
                          printf("4. Update the list of generic servers for an account\n");
                          printf("5. Update Account Information\n");
                          printf("6. Update Service Information\n");
                          printf("q. To quit the menu\n");
                          memset(menu,'\0',sizeof(menu));
                           getUinput(menu,sizeof(menu));
                  }while(strlen(menu)==0);

                  if (flagAuthen ==1 && strncmp(menu,"q",1))
                   {
                          userInfo.name = (char*)malloc(sizeof(char)*64);
                  userInfo.word = (char*)malloc(sizeof(char)*64);
                  userInfo.address = (char*)malloc(sizeof(char)*1);
                  userInfo.phoneNo = (char*)malloc(sizeof(char)*1);
                  userInfo.email = (char*)malloc(sizeof(char)*1);
```

```
                        memset(userInfo.address,'\0',1);
                        memset(userInfo.phoneNo,'\0',1);
                        memset(userInfo.email,'\0',1);

                        docreate = getUserInfo(&userInfo,0);
                if(debug)
                        {
                                printf("Before making rpc call\n");
                                printf("Name *%s*\n",userInfo.name);
                 printf("Word *%s*\n",userInfo.word);
                        }
                        if(!docreate)
                        {
                                time(&startexectime);
                                startcputime=clock();
                                authen(rh,&userInfo,busid,&status);
                                printf("\n");
                          time(&endexectime);
                          endcputime=clock();
             if(debug)printf("Time elapsed to authenticate a user cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);
                                if(status==0)
                 {
                        printf("Account authenticated with bus id %s\n",busid);

                                        flagAuthen=0;
                 }
                 else
                 {
                        printf("%s\n",busid);
                                        printf("\nPress ENTER to go back to the main menu\n");
                        getUinput(dummy,32);
                                }
                        }


                        free(userInfo.name);
                free(userInfo.word);
                free(userInfo.address);
                free(userInfo.phoneNo);
                free(userInfo.email);
                  }

                  if(!strncmp(menu,"1",1) && flagAuthen==0)
                 {

                        genericServers.first=0;
                genericServers.size=1;
                strcpy(genericServers.array,"");
```

```
                    getGenericServers(rh,busid,&genericServers,&status);

                    if(status==0)
                    {
                                strcpy(backupGeneric,genericServers.array);

                     if(strncmp(genericServers.array,"ERROR",5))
                     {
                         printf("Following genericServers are available\n");
                         temp=strtok(genericServers.array,",");

                         while(temp!=NULL)
                         {
                             printf("%s\n",temp);
                             temp=strtok(NULL,",");
                         }
                     }
                     else
                     {
                         printf("%s",genericServers.array);
                     }
                    }
                    else
                    {
                           printf("%s",genericServers.array);
                    }

                    /*Add or delete genericServers */
                    printf("\n\nWould like to add or delete from generic Servers List? If yes
enter 'y' or enter 'n'\n");
                    getUinput(dummy,32);

                    if(!strncmp(dummy,"y",1))
                    {
                           standby=(char **)malloc(sizeof(char**));

                    docreate = getUpdateDet("GENERIC",standby,&type);

                  if(debug)
              {
                     printf("Before making rpc call\n");
                 printf("generic Servers *%s*\n",*standby);
                    printf("type = %d\n",type);
                           }
            if(!docreate)
            {
                    standbypaths=(idl_char *)strdup(*standby);
                    serviceName=(idl_char*)strdup("GENERIC");
                    updateServers(rh,&type,serviceName,standbypaths,busid,statusString);
                    printf("\n");
```

```
                printf("%s\n",statusString);

                                /* Read the updated list */
                                genericServers.first=0;
                      genericServers.size=1;
                    strcpy(genericServers.array,"");

                    getGenericServers(rh,busid,&genericServers,&status);
                                if(status==0)
                 {
                 strcpy(backupGeneric,genericServers.array);
                                }
                        }

        free(*standby);
        free(standby);
                }
                genpaths=(char **)malloc(sizeof(char**));
                standby=(char **)malloc(sizeof(char**));


                serviceDet.name = (char*)malloc(sizeof(char)*64);
                serviceDet.description = (char*)malloc(sizeof(char)*256);
                serviceDet.keywords = (char*)malloc(sizeof(char)*64);
                serviceDet.DocumentUrl = (char*)malloc(sizeof(char)*256);

      memset(serviceDet.name,'\0',64);
      memset(serviceDet.description,'\0',256);
      memset(serviceDet.keywords,'\0',64);
      memset(serviceDet.DocumentUrl,'\0',256);

                docreate = getServiceDet(service,&serviceDet,genpaths,standby,&type);

                if(debug)
                {
                        printf("Before making rpc call\n");
                        printf("Paths *%s*\n",*genpaths);
        printf("standby *%s*\n",*standby);
                printf("service *%s*\n",service);
                        printf("Type = %d\n",type);
                }

                if(!docreate && type!=5)
                {
                        resultback.first=0;
        resultback.size=1;
        strcpy(resultback.array,"");
                        paths=(idl_char *)strdup(*genpaths);
                        standbypaths=(idl_char *)strdup(*standby);
                        serviceName=(idl_char*)strdup(service);
                        time(&startexectime);
```

```
                                startcputime=clock();


        serviceAction(rh,serviceName,&serviceDet,paths,standbypaths,busid,&type,&resultback,
&status);
                                printf("\n");
                                time(&endexectime);
                                endcputime=clock();
         if(debug)printf("Time elapsed to execute a service action cpu %ld exec %ld
\n",endcputime-startcputime,endexectime-startexectime);


                                if(status==-1)
                                        printf("%s\n",resultback.array);
                                else if(status==0)
                                {
                                        if(type==CREATE)
                                        {
                                                printf("Web Service group created with the
following responses for each generic path\n");
                                        }
                                        else
                                                printf("The following responses were received
for each generic path\n");

                                        if(type==CREATE || type==START)
                                                printf("NOTE: If response is a http path, it is the
access point of the service started\n");

        printTwoCol(*genpaths,resultback.array,"Generic Paths","Responses");

                                } else if(status==1)
                                {
                                        printf("The following responses were received for each
generic path\n");
                                        if(type==CREATE || type==START)
                                printf("NOTE: If response is a http path, it is the path of the
service started\n");
                                        printTwoCol(*genpaths,resultback.array,"Generic
Paths","Responses");
                        printf("But there was an error in updating the database. Contact
Administrator\n");
                                        if(type==CREATE)
                                                printf("Service group created not reflected in the
database\n");
                                } else if(status==-2)
                                {
                                        if(type==CREATE)
                                        printf("No service(s) started, hence group not
created\n");
                                        else if(type==START)
                                        printf("No serivce(s) was successfully started\n");
                                        printf("The following responses were received for each
generic path\n");
                        printTwoCol(*genpaths,resultback.array,"Generic Paths","Responses");

                                }
                        }

                        if(!docreate && type==5)
                        {
                                serviceName=(idl_char*)strdup(service);
                                time(&startexectime);
                                startcputime=clock();
                                deleteService(rh,serviceName,busid,statusString);
                                printf("%s\n",statusString);
                                time(&endexectime);
                                endcputime=clock();
                      if(debug)printf("Time elapsed to delete a web service cluster cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);
                        }

                        printf("\nPress ENTER to go back to the main menu\n");
                getUinput(dummy,32);

                        free(*genpaths);
                        free(*standby);
                        free(genpaths);
                        free(standby);

                        free(serviceDet.name);
                free(serviceDet.description);
                free(serviceDet.keywords);
                free(serviceDet.DocumentUrl);
                  }

                  if(!strncmp(menu,"2",1)&& flagAuthen ==0)
           {
                standby=(char **)malloc(sizeof(char**));

                        genericServers.first=0;
                genericServers.size=1;
                strcpy(genericServers.array,"");

                getGenericServers(rh,busid,&genericServers,&status);

                if(status==0)
                {
                        strcpy(backupGeneric,genericServers.array);
                }
                else
                {
                    printf("%s",genericServers.array);
                }


                        docreate = getUpdateDet(service,standby,&type);

                        if(debug)
                        {
                                printf("Before making rpc call\n");
                  printf("standby *%s*\n",*standby);
                  printf("service *%s*\n",service);
                        }
                if(!docreate)
                {
                    standbypaths=(idl_char *)strdup(*standby);
                    serviceName=(idl_char*)strdup(service);
                    time(&startexectime);
                                startcputime=clock();

        updateServers(rh,&type,serviceName,standbypaths,busid,statusString);
                                printf("\n");
                                time(&endexectime);
                                  endcputime=clock();
            if(debug)printf("Time elapsed to update servers cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);
                                printf("%s\n",statusString);
                }

                        printf("\nPress ENTER to go back to the main menu\n");
                getUinput(dummy,32);

                        free(*standby);
                free(standby);

            }

              if(!strncmp(menu,"3",1) && flagAuthen == 0)
            {

                        userInfo.name = (char*)malloc(sizeof(char)*64);
                userInfo.word = (char*)malloc(sizeof(char)*64);
                userInfo.address = (char*)malloc(sizeof(char)*256);
                userInfo.phoneNo = (char*)malloc(sizeof(char)*64);
                userInfo.email = (char*)malloc(sizeof(char)*64);

                        serviceNames.first=0;
                serviceNames.size=1;
                strcpy(serviceNames.array,"");

                        serviceIds.first=0;
                serviceIds.size=1;
                strcpy(serviceIds.array,"");
                        time(&startexectime);
                        startcputime=clock();
                        getBusidInfo(rh,busid,&userInfo,&serviceNames,&serviceIds,&status);
                                printf("\n");
                 time(&endexectime);
                          endcputime=clock();
          if(debug)printf("Time elapsed to get information for a give account cpu %ld exec %ld
\n",endcputime-startcputime,endexectime-startexectime);
                        if(status==2)
                        {
                                printf("%s\n",serviceNames.array);

                                 printf("\nPress ENTER to go back to the main menu\n");
                 getUinput(dummy,32);
                        }
                        else if (status==1)
                        {
                                printf("The information for the given account is:\n");
                                printf("Address: %s\n",userInfo.address);
                                printf("Email: %s\n",userInfo.email);
                                printf("Phone number: %s\n",userInfo.phoneNo);
                                printf("\n%s\n",serviceNames.array);

                                 printf("\nPress ENTER to go back to the main menu\n");
                 getUinput(dummy,32);
                        }
                        else if (status==0)
                        {
                                printf("The information for the given account is:\n");
                    printf("Address: %s\n",userInfo.address);
                    printf("Email: %s\n",userInfo.email);
                    printf("Phone number: %s\n",userInfo.phoneNo);

                                printf("\nThe following services were found\n");
                                printTwoCol(serviceNames.array,serviceIds.array,"Service
Names","Service Ids");

                          do{

                                printf("\nEnter the name of the service to get the Info for\n");
                                do{
                        printf("Enter the service name:\n");
                        memset(service,'\0',32);
                                        getUinput(service,32);
                        }while(strlen(service)==0);

                                replicas.first=0;
                 replicas.size=1;
                 strcpy(replicas.array,"");
```

```
                                    replicatimes.first=0;
                            replicatimes.size=1;
                            strcpy(replicatimes.array,"");

                                    standbyservers.first=0;
                            standbyservers.size=1;
                            strcpy(standbyservers.array,"");
                                    time(&startexectime);
                                    startcputime=clock();

        getServiceInfo(rh,service,busid,&replicas,&replicatimes,&standbyservers,&status);
                                    printf("\n");
                                    time(&endexectime);
                                      endcputime=clock();
         if(debug)printf("Time elapsed to get Service Information cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);
                                    if(status==1)
                              {
                              printf("%s\n",replicas.array);

                           }
                                    else
                                    {
                                            printf("The Info for the service %s is:\n",service);
                                            serviceCount=0;
                                            if(debug)
                                            {
                                                    printf("Replicas *%s*\n",replicas.array);
                                                    printf("TimeStamps
*%s*\n",replicatimes.array);
                                                    printf("Back-up *%s*\n",standbyservers.array);
                                            }
                                            if(strncmp(replicas.array,"ERROR",5))
                                            {
                                                    printf("Services running at:\n");

        printTwoCol(replicas.array,replicatimes.array,"Replicas","Time started");
                                            }
                                            else printf("%s\n",replicas.array);


                                            if(strncmp(standbyservers.array,"ERROR",5))
                                            {
                                                    printf("Back-up servers for the service are:\n");

                                            if((temp=strtok(standbyservers.array,","))!=NULL)
                                            {
                                                    printf("%s\n",temp);
                                                    }

                                            while((temp=strtok(NULL,","))!=NULL)
```

```
                                        {
                                                  printf("%s\n",temp);
                                        }
                                        }
                                        else printf("%s\n",standbyservers.array);

                              }

                              printf("\nEnter m to get info about more services or enter b to go
back to the main menu\n");

                    memset(service,'\0',32);
                                        getUinput(service,32);
                        }while(!strncmp(service,"m",1));

                    }

                    free(userInfo.name);
          free(userInfo.word);
          free(userInfo.address);
          free(userInfo.phoneNo);
          free(userInfo.email);


            }

            if(!strncmp(menu,"4",1)&& flagAuthen ==0)
    {
                    standby=(char **)malloc(sizeof(char**));

                    genericServers.first=0;
          genericServers.size=1;
          strcpy(genericServers.array,"");

          getGenericServers(rh,busid,&genericServers,&status);

          if(status==0)
          {
                    strcpy(backupGeneric,genericServers.array);
                    }
                    /*else
                    {
                              printf("%s\n",genericServers.array);
                    }*/
               docreate = getUpdateDet("GENERIC",standby,&type);

              if(debug)
               {
                    printf("Before making rpc call\n");
                    printf("generic Servers *%s*\n",*standby);
                    printf("type = %d\n",type);
```

```
                        }
                        if(!docreate)
                        {
                                standbypaths=(idl_char *)strdup(*standby);
                                serviceName=(idl_char*)strdup("GENERIC");
                                time(&startexectime);
                                                        startcputime=clock();

                updateServers(rh,&type,serviceName,standbypaths,busid,statusString);
                                printf("\n");
                                time(&endexectime);
                                                         endcputime=clock();
                 if(debug)printf("Time elapsed to update serverss cpu %ld exec %ld\n",endcputime-
        startcputime,endexectime-startexectime);
                                                        printf("%s\n",statusString);

                                /* Read the updated list */
                                genericServers.first=0;
                                genericServers.size=1;
                                strcpy(genericServers.array,"");

                                getGenericServers(rh,busid,&genericServers,&status);
                                if(status==0)
                                {
                                        strcpy(backupGeneric,genericServers.array);
                                }
                        }

                        free(*standby);
                        free(standby);
                                printf("\nPress ENTER to go back to the main menu\n");
                    getUinput(dummy,32);
                }

                     if(!strncmp(menu,"5",1) && flagAuthen == 0)
                     {

                                userInfo.name = (char*)malloc(sizeof(char)*64);
                    userInfo.word = (char*)malloc(sizeof(char)*64);
                    userInfo.address = (char*)malloc(sizeof(char)*256);
                    userInfo.phoneNo = (char*)malloc(sizeof(char)*64);
                    userInfo.email = (char*)malloc(sizeof(char)*64);

                    docreate = getUpdateUserInfo(&userInfo);
                    if(debug)
                                {
                                        printf("Before making rpc call\n");
                                        printf("Name *%s*\n",userInfo.name);
                     printf("Word *%s*\n",userInfo.word);
                     printf("Add *%s*\n",userInfo.address);
                     printf("Phone *%s*\n",userInfo.phoneNo);
                printf("Email *%s*\n",userInfo.email);
                        }
                if(!docreate)
                {
                                time(&startexectime);
                                startcputime=clock();
                     updateAccount(rh,&userInfo,busid,statusString);
                                time(&endexectime);
                                 endcputime=clock();
           if(debug)printf("Time elapsed to update account information cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);
                                printf("\n");

                }

                        printf("%s\n",statusString);

                        printf("\nPress ENTER to go back to the main menu\n");
                getUinput(dummy,32);
                   free(userInfo.name);
                free(userInfo.word);
                free(userInfo.address);
                free(userInfo.phoneNo);
                free(userInfo.email);
                  }

                 if(!strncmp(menu,"6",1) && flagAuthen == 0)
                 {

                        serviceDet.name = (char*)malloc(sizeof(char)*64);
                serviceDet.description = (char*)malloc(sizeof(char)*256);
                serviceDet.keywords = (char*)malloc(sizeof(char)*64);
                serviceDet.DocumentUrl = (char*)malloc(sizeof(char)*256);

                memset(serviceDet.name,'\0',64);
                memset(serviceDet.description,'\0',256);
                memset(serviceDet.keywords,'\0',64);
                memset(serviceDet.DocumentUrl,'\0',256);

                        docreate = getUpdateServiceDet(service,&serviceDet);

                         if(!docreate)
                {
                                time(&startexectime);
                                startcputime=clock();
                                updateServiceInfo(rh,service,&serviceDet,busid,statusString);
                                time(&endexectime);
                             endcputime=clock();
           if(debug)printf("Time elapsed to update service information cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);
                                printf("\n");
                        }

                        printf("%s\n",statusString);

                        printf("\nPress ENTER to go back to the main menu\n");
                   getUinput(dummy,32);

                        free(serviceDet.name);
                        free(serviceDet.description);
                        free(serviceDet.keywords);
                        free(serviceDet.DocumentUrl);
                    }


        }while(strncmp(menu,"q",1));
        exit(0);
}  /* end main() */


int getUserInfo(accountInfo *userInfo, int func)
{

   char answer[8];
   int i,value =0;
   char path[128];
      int len;

        memset(answer,'\0',8);

   do{

                    memset(userInfo->name,'\0',64);
                    do{
                        printf("Enter the username:\n");
                        getUinput(userInfo->name,64);
                        if(debug)printf("*%s*\n",userInfo->name);
                     }while(strlen(userInfo->name)==0);


                    memset(userInfo->word,'\0',64);
                    do{
                        strcpy(userInfo->word,getpass("Enter the password:"));
                        if(debug)printf("*%s*\n",userInfo->word);
                      }while(strlen(userInfo->word)==0);
                        code(userInfo->word);

                    if(func==1)
                    {
                        memset(userInfo->address,'\0',256);
                        do{
                        printf("Enter the Address: (Not more than 200 characters)\n");
                        getUinput(userInfo->address,256);
                                if(debug)printf("*%s*\n",userInfo->address);
                }while(strlen(userInfo->address)==0);
                        for(i=0;i<strlen(userInfo->address);++i)
                                if(userInfo->address[i]==',')userInfo->address[i]='.';
                        memset(userInfo->phoneNo,'\0',64);
                        do{
                        printf("Enter the Phone number:\n");
                        getUinput(userInfo->phoneNo,64);
                                if(debug)printf("*%s*\n",userInfo->phoneNo);
                }while(strlen(userInfo->phoneNo)==0);
                        memset(userInfo->email,'\0',64);

                        do{
                        printf("Enter the email address:\n");
                        getUinput(userInfo->email,64);
                                if(debug)printf("*%s*\n",userInfo->email);
                }while(strlen(userInfo->email)==0);


                }

                do{
                        printf("If the information entered is correct enter 'y' or enter 'n' to re-enter
the information\n");
                        printf("To cancel enter 'c'\n");
                        getUinput(answer,sizeof(answer));
                 }while(strncmp(answer,"n",1) && strncmp(answer,"c",1) &&
strncmp(answer,"y",1));

                if(debug)printf("*%s*\n",answer);
        }while(strncmp(answer,"y",1) && strncmp(answer,"c",1));

        if(!strncmp(answer,"c",1))
                value=1;
        if(debug)printf("%d",value);
        return value;
}

int getUpdateDet(char *service, char **standby,int *type)
{

         char answer[8];
     int value =0;
     char path[128];
     int len;
        char genericPaths[1024];

        do{
          memset(answer,'\0',8);
          memset(genericPaths,'\0',1024);
                *standby=(char *)malloc(sizeof(char));
          *(standby[0])='\0';

                if(strncmp(service,"GENERIC",7))
                {
                        do{
                printf("Enter the service name:\n");
                getUinput(service,32);
                if(debug)printf("*%s*\n",service);
                }while(strlen(service)==0);
                }
                do{
               printf("Enter one of the following options:\n");
               printf("1. Add Servers\n");
               printf("2. Delete Servers\n");
               getUinput(answer,8);
               if(debug)printf("*%s*\n",answer);
             }while(strlen(answer)==0);

          *type=atoi(answer);

                if(strncmp(service,"GENERIC",7) || *type==2)
                {
                                getPathsfromUser(genericPaths);
                    *standby=(char *)malloc(sizeof(char)*(strlen(genericPaths)+1));
                    strcpy(*standby,genericPaths);
                    if(debug)printf("Gpaths *%s*\n",*standby);
                }
                else
                {
                        len = 0;
                printf("Enter the access point(s) of the genericSoapServer (Enter 'd' to continue
to the next step)\n");
                do{
                do{
                        memset(path,'\0',128);
                        printf("Enter the access point\n");
                        getUinput(path,128);
                }while(strlen(path)==0);

                if(strncmp(path,"d",1))
                {
                        len = strlen(path) + len +1;
                        *standby=(char *)realloc((void*)*standby,sizeof(char)*len);
                        strcat(*standby,path);
                        strcat(*standby,",");
                }
                }while(strncmp(path,"d",1));
                if(debug)printf("spaths *%s*\n",*standby);
                }
```

```
                    do{
                  printf("If the information entered is correct enter 'y' or enter 'n' to re-enter the
information\n");
                  printf("To cancel enter 'c'\n");
                  getUinput(answer,sizeof(answer));
                }while(strncmp(answer,"n",1) && strncmp(answer,"c",1) &&
strncmp(answer,"y",1));

                    if(!strncmp(answer,"n",1))
                          free(*standby);

    }while(strncmp(answer,"y",1) && strncmp(answer,"c",1));

    if(!strncmp(answer,"c",1))
         value=1;
    if(debug)printf("%d",value);
       return value;

}
int getServiceDet(char *service,serviceInfo *serviceDet, char **genpaths,char **standby,int
*type)
{

       char answer[8];
       int i,value =0;
       char path[128];
       char genericPaths[1024];
       int len;

       do{
             memset(service,'\0',32);
       memset(answer,'\0',8);
         memset(genericPaths,'\0',1024);
             *standby=(char *)malloc(sizeof(char));
         *genpaths=(char *)malloc(sizeof(char));
             *(standby[0])='\0';
             *(genpaths[0])='\0';


             do{
               printf("Enter the service name:\n");
               getUinput(service,32);
               if(debug)printf("*%s*\n",service);
            }while(strlen(service)==0);

             do{
               system("clear");
                     printf("Enter one of the following options:\n");
               printf("1. Start a webservice cluster\n");
               printf("2. Start a webservice instance in an existing cluster\n");
```

```
                printf("3. Stop a webservice instance in an existing cluster\n");
                printf("4. Get Status of a webservice instance in an existing cluster\n");
                printf("5. Delete a webservice cluster\n");
                        printf("6. Go back to the main menu\n");
                        getUinput(answer,8);
                if(debug)printf("*%s*\n",answer);
            }while(strlen(answer)==0);

                 *type=atoi(answer);
         if(strncmp(answer,"6",1))
         {
                        if(*type!=5)
                        {
                                printf("Enter the access point(s) of the genericSoapServer\n");
                                getPathsfromUser(genericPaths);
                                *genpaths=(char
*)malloc(sizeof(char)*(strlen(genericPaths)+1));
                                strcpy(*genpaths,genericPaths);
                                if(debug)printf("Gpaths *%s*\n",*genpaths);
                        }

                        if(*type==1)
                        {

                                printf("Enter the details for the web service\n");
                        memset(serviceDet->name,'\0',64);
                                do{
                        printf("Enter the name to be entered in the Web service Registry:\n");
                        getUinput(serviceDet->name,64);
                        if(debug)printf("*%s*\n",serviceDet->name);
                        }while(strlen(serviceDet->name)==0);


                        memset(serviceDet->description,'\0',256);
                        do{
                        printf("Enter the description\n");
                        getUinput(serviceDet->description,256);
                                        if(debug)printf("*%s*\n",serviceDet->description);
                        }while(strlen(serviceDet->description)==0);
                                for(i=0;i<strlen(serviceDet->description);++i)
                                        if(serviceDet->description[i]=='\'')serviceDet-
>description[i]='.';

                                memset(serviceDet->keywords,'\0',64);
                        do{
                        printf("Enter the keywords (separated by commas)\n");
                        getUinput(serviceDet->keywords,64);
                        if(debug)printf("*%s*\n",serviceDet->keywords);
                  }while(strlen(serviceDet->keywords)==0);

                                memset(serviceDet->DocumentUrl,'\0',256);
```

```
                    do{
                    printf("Enter the Document URL\n");
                    getUinput(serviceDet->DocumentUrl,256);
                    if(debug)printf("*%s*\n",serviceDet->DocumentUrl);
                }while(strlen(serviceDet->DocumentUrl)==0);

                    memset(genericPaths,'\0',1024);
                    printf("Enter the access point(s) of the back-up genericSoapServer \n");
                            getPathsfromUser(genericPaths);
                  *standby=(char *)malloc(sizeof(char)*(strlen(genericPaths)+1));
                  strcpy(*standby,genericPaths);
                            if(debug)printf("spaths *%s*\n",*standby);
                    }
                 do{
                printf("If the information entered is correct enter 'y' or enter 'n' to re-enter the
information\n");
                printf("To cancel enter 'c'\n");
                getUinput(answer,sizeof(answer));
            }while(strncmp(answer,"n",1) && strncmp(answer,"c",1) && strncmp(answer,"y",1));

                if(!strncmp(answer,"n",1))
                    {
                            free(*standby);
                            free(*genpaths);
                    }
        }/*if ends*/
        else
        strcpy(answer,"c");

    }while(strncmp(answer,"y",1) && strncmp(answer,"c",1));

    if(!strncmp(answer,"c",1))
          value=1;
    if(debug)printf("%d",value);
    return value;


}

int getUpdateUserInfo(accountInfo *userInfo)
{

  char answer[8];
  int i,value =0;
    memset(answer,'\0',8);

          do{

            memset(userInfo->word,'\0',64);
            printf("Enter the new password or just press enter to skip:\n");
            getUinput(userInfo->word,64);
            if(debug)printf("*%s*\n",userInfo->word);
                  code(userInfo->word);
                  memset(userInfo->address,'\0',256);
            printf("Enter the new address or just press enter to skip:\n");
            getUinput(userInfo->address,256);
            if(debug)printf("*%s*\n",userInfo->address);
                  for(i=0;i<strlen(userInfo->address);++i)
                      if(userInfo->address[i]==',')userInfo->address[i]='.';
                  memset(userInfo->phoneNo,'\0',64);
            printf("Enter the new Phone number or just press enter to skip:\n");
            getUinput(userInfo->phoneNo,64);
            if(debug)printf("*%s*\n",userInfo->phoneNo);

                  memset(userInfo->email,'\0',64);
            printf("Enter the new email or just press enter to skip:\n");
            getUinput(userInfo->email,64);
            if(debug)printf("*%s*\n",userInfo->email);

                  do{
                 printf("If the information entered is correct enter 'y' or enter 'n' to re-enter the
information\n");
                 printf("To cancel enter 'c'\n");
                 getUinput(answer,sizeof(answer));
              }while(strncmp(answer,"n",1) && strncmp(answer,"c",1) &&
strncmp(answer,"y",1));

       if(debug)printf("*%s*\n",answer);
       }while(strncmp(answer,"y",1) && strncmp(answer,"c",1));

          if(!strncmp(answer,"c",1))
            value=1;
       if(debug)printf("%d",value);
       return value;

}

int getUpdateServiceDet(char *service,serviceInfo *serviceDet)
{

   char answer[8];
   int value =0;
      memset(answer,'\0',8);

       do{


                  do{
                  memset(service,'\0',32);
            printf("Enter the name of the service:\n");
            getUinput(service,32);
            if(debug)printf("*%s*\n",service);
                  }while(strlen(service)==0);
            memset(serviceDet->name,'\0',64);
            printf("Enter the new name to be entered in the WSR or just press enter to skip:\n");
            getUinput(serviceDet->name,64);
            if(debug)printf("*%s*\n",serviceDet->name);

            memset(serviceDet->keywords,'\0',64);
            printf("Enter the new keywords (separated by commas) or just press enter to skip:\n");
            getUinput(serviceDet->keywords,64);
            if(debug)printf("*%s*\n",serviceDet->keywords);

            memset(serviceDet->description,'\0',256);
            printf("Enter the new description or just press enter to skip:\n");
            getUinput(serviceDet->description,256);
            if(debug)printf("*%s*\n",serviceDet->description);

            memset(serviceDet->DocumentUrl,'\0',256);
            printf("Enter the new Document URL or just press enter to skip:\n");
            getUinput(serviceDet->DocumentUrl,64);
            if(debug)printf("*%s*\n",serviceDet->DocumentUrl);

            do{
                printf("If the information entered is correct enter 'y' or enter 'n' to re-enter the
information\n");
                printf("To cancel enter 'c'\n");
                getUinput(answer,sizeof(answer));
             }while(strncmp(answer,"n",1) && strncmp(answer,"c",1) &&
strncmp(answer,"y",1));

        if(debug)printf("*%s*\n",answer);
        }while(strncmp(answer,"y",1) && strncmp(answer,"c",1));

           if(!strncmp(answer,"c",1))
             value=1;
        if(debug)printf("%d",value);
        return value;

}

void printTwoCol(char *col1,char* col2,char *title1,char *title2)
{
                char *temp,*temp1,*value,*value1;
                int count=0;

                printf("\n  %-85s %s\n\n",title1,title2);
                                        while((temp = strrchr(col1,','))!=NULL)
                        {
                                                printf("%d.",++count);
                            *temp='\0';
                            if((temp=strrchr(col1,','))!=NULL)
                            {
```

```
                        value=temp+1;
                        printf("%-80s",value);
                    }
                    else
                        printf("%-80s",col1);

                    temp1 = strrchr(col2,',');
                    *temp1='\0';
                    if((temp1=strrchr(col2,','))!=NULL)
                    {
                        value1=temp1+1;
                        printf("\t%s\n",value1);
                    }
                    else
                        printf("\t%s\n",col2);

                }
}



/***********************************************************************
 *
 * Author:   G.K.Springer
 * Date:     August 1994
 * Place:    Computer Science Department, University of Missouri-Columbia
 * Node:     condor.cs.missouri.edu
 *
 * Purpose:  Allocation Procedure for buffer space in the Pipe.
 *
 * Function: Initialize and allocate the stuff needed for a DCE Pipe.
 *
 * Update Log:
 *
 ***********************************************************************/

void client_alloc(state, bsize, buf, bcount)  /* allocation for a pipe */
  pipe_state    *state;   /* corrdinates pipe procedure calls */
  idl_ulong_int bsize;    /* desired buffer size in bytes */
  idl_char      **buf;    /* allocated buffer */
  idl_ulong_int *bcount;  /* allocated buffer size in bytes */
{
idl_char  client_buffer[BUF_SIZE];
  *buf = client_buffer;   /* assign address of the buffer */
  if (bsize <= BUF_SIZE)
     *bcount = bsize;  /* give DCE the buffer size it wants */
  else
     *bcount = BUF_SIZE;  /* otherwise give DCE what we have to give */
  return;
}  /* end client_alloc() */
```

```
/*********************************************************************
 *
 * Author:   G.K.Springer
 * Date:     August 1994
 * Place:    Computer Science Department, University of Missouri-Columbia
 * Node:     condor.cs.missouri.edu
 *
 * Purpose:  This is the application dependent code for the RPC Pipe Pull
 *           procedure for an input pipe.
 *
 * Function: Open a file on the server side and read the data from the
 *           file and put what is read into the pipe.  Data is put into
 *           the pipe in "chunks" until EOF.
 *
 * Update Log:
 *
 *********************************************************************/


void in_pull(state, buf, esize, ecount)  /* input pipe uses pull procedure */
  pipe_state     *state;  /* coordinates the pipe procedure calls */
  idl_char       *buf;    /* buffer of data pulled */
  idl_ulong_int  esize;   /* max element count in buffer */
  idl_ulong_int  *ecount; /* actual element count in buffer */
{
/* For this application, open local source file if not open already */
  if (state->filehandle == -1) {
    state->filehandle = open(state->filename, O_RDONLY);
    if (state->filehandle == -1) {
      fprintf(stderr, "Cannot open %s for read\n", state->filename);
      exit(0);
      }
    } /* if file not open */

/* Process a buffer for the application */
  *ecount = read(state->filehandle, buf, esize);  /* read a buffer full */

/* to signal end of data, pull procedure must set the count to 0 */
  if (*ecount == 0) { /* end of data, do application cleanup */
    close(state->filehandle);  /* close the input file */
    }
  return;
}  /* end in_pull() */

/*********************************************************************
 *
 * Author:   G.K.Springer
 * Date:     August 1994
 * Place:    Computer Science Department, University of Missouri-Columbia
 * Node:     condor.cs.missouri.edu
 *
 * Purpose:  This is the application dependent code for the RPC Pipe Push
 *           procedure for an output pipe.
 *
 * Function: Open a file on the server side and write the data from the
 *           pipe to the file located on the server.  Data is pushed by
 *           the client into the pipe in "chunks" until EOF.  The server
 *           takes this data and writes a copy of the data into a file on
 *           the server host.
 *
 * Update Log:
 *
 ***********************************************************************/
void out_push(state, buf, ecount)  /* output pipe uses push procedure */
  pipe_state     *state;  /* coordinates the pipe procedure calls */
  idl_char       *buf;    /* buffer of data pushed */
  idl_ulong_int  ecount; /* actual element count in buffer */
{
/* For this application, open local target file if not open already */
  if (state->filehandle == -1) {
    if (ecount <= 0) /* if first buffer is empty, don't do anything */
      return;
    state->filehandle = open(state->filename,
                      O_CREAT | O_TRUNC | O_WRONLY, 0644);
    if (state->filehandle == -1) {
      fprintf(stderr, "Cannot open %s for write\n", state->filename);
      exit(0);
      }
    fchmod(state->filehandle, 0644); /* set file permissions */
    } /* if file not open */


/* Process a buffer for the application */
  if (ecount == 0) { /* end of data, do application cleanup */
    close(state->filehandle);  /* close the input file */
    state->filehandle = -1;    /* reset handle to a closed file */
    }
  else
    write(state->filehandle, buf, ecount);
  return;
}  /* end out_push() */


/***********************************************************************
 * Name : code
 * Purpose: To encrypt a word.
 * Function: To use the system call "crypt" to encode the string.
 * Returns:  copies the encoded string in the buffer sent by the calling function.
 ***********************************************************************/

void code(char *session_key)
```

```
{
        char *res;
        char session_salt[3];
        session_salt[0]=session_key[strlen(session_key)-1];
        session_salt[1]=session_key[0];
        session_salt[2]='\0';
        res = crypt(session_key, session_salt);
        strcpy(session_key,res);
}

void getPathsfromUser(char *paths)
{

             char *temp;
             char localcopy[1024];
             int i,count=0;
             char answer[32];
             char *pathGeneric[32];
             int noGenPath=0;

             memset(answer,'\0',32);

             strcpy(localcopy,backupGeneric);

             /* Parse the comma separated generic paths to form an array of Strings */
        if(strlen(localcopy)!=0)
        {

               temp=strtok(localcopy,",");

               while(temp!=NULL)
               {
                      pathGeneric[noGenPath]=(char *)malloc(sizeof(char)*(strlen(temp)+1));
                      strcpy(pathGeneric[noGenPath],temp);
                      noGenPath++;
                                  temp=strtok(NULL,",");
                        }

        }
                       printf(" %d Following genericServers are available\n",noGenPath);
                       for(i=0;i<noGenPath;++i)
                       {
                                  printf("%d. %s\n",i+1,pathGeneric[i]);
                       }

                       printf("Enter the numbers of the servers separated by commas. e.g 1,2\n");
                       getUinput(answer,32);

                       temp=strtok(answer,",");

               while(temp!=NULL)
```

```
                {
                        strcat(paths,pathGeneric[atoi(temp)-1]);
                        strcat(paths,",");
                                        temp=strtok(NULL,",");
                }

                           for(i=0;i<noGenPath;++i)
                                {
                          free(pathGeneric[i]);
                          }

                                        free(pathGeneric);
}
```

Ws_admin

1) Ws_admin.c

```
#pragma options showinc source
/************************************************************************
 * DCE Program ws_admin:  ws_admin.c - Client Application Program
 *
 * Author:   Khambati Abdulkadar
 * Date:     June 2004
 * Location: CECS Department, University of Missouri-Columbia
 * Node:     darwin.rnet.missouri.edu
 *
 * Purpose:  DCE client for Web Service Manager
 *
 * Function: It calls the procedures of the Web Service Manager
 *
 *
 * Syntax:   This program is invoked with the command line:
 *             ws_admin <server_host>
 *          where:
 *           server_host - identifies the host where server is running
 * Example:  ws_admin darwin
 ************************************************************************/

#include <wsmclient.h>
#include <database.h>
int findserver();    /* Utility function to look up the name service and get the bindings for the
registered servers */
int locserver();    /*  Utility function to Locate an active server on a designated host machine.*/
void client_alloc(), in_pull(),out_push();  /* from stub code */
int getUserInfo(accountInfo *,int);
void printTwoCol(char*,char*,char*,char*);
void code(char *);

char tmpstr[256];
char *rdptr;
#define getUinput(inp, size) {  memset((inp), 0, (size)); \
  rdptr = fgets(tmpstr, 256, stdin); \
  if (rdptr != NULL) { if (strlen(rdptr) > (size)) *(rdptr+(size)-1)='\0'; \
      strcpy((inp),rdptr); \
      if(strrchr((inp),'\n')) \
        inp[strlen(inp)-1]='\0'; \
     } \
  else strcpy((inp), "NULL"); \
  }

#define REQUESTFILE "/usr/users/aakwv5/webservices/testexamples/WSM/requestAccount"
int debug;

#ifdef DEBUGL
```

```
debug=1;
#else
debug=0;
#endif


main(argc, argv)
  int argc;
  char *argv[];
{
  int num; /* to hold the return value from locserver function */
      unsigned32 st;  /* To hold DCE status code from locserver */
      unsigned32 status;
      int num_entry;
      int          nsrv;  /* number of servers in demo run */
      int          index[MaxSrv+1];  /* Index to servers */
      int          proto[MaxSrv+1]; /* Index to handles we use */
      int          docreate;
      unsigned32 type;
      char    *host,         /* hostname for server */
      *objstr;       /*object string value */
      rpc_binding_handle_t rh;        /* To hold the binding handle to communicate with the server
*/
      rpc_binding_vector_t *results;
      char       group_name[NMLEN];
      pipe_state state;        /* state of the pipe */
      pipe_c     data;         /* a pipe structure is allocated */
      char       lcl_source[100];  /* to hold the name of the file to be sent through pipe  */
      char       lcl_target[100];  /* to hold the name of the target file for file received through
pipe*/
      file_spec  cbind_h;      /* customized binding handle */
      int     i,j;
      conarray  userNames,userIds;
      idl_char   busid[64];
      accountInfo userInfo;
      char      menu[8];
          char buffer[256];
      char *temp;
          char filename[64];
          char command[64];
      FILE *fp,*fp1;
          clock_t   clock(),startcputime,endcputime;
      time_t  startexectime,endexectime;

      time(&startexectime);
      startcputime = clock();

      host = strdup(argv[1]); /* move Server hostname */
      objstr = strdup(OBJSTR); /* copy the Server's DCE object UUID */
      strcpy(group_name, GRPNAME);  /* identify the group desired */
```

```
#ifndef NSREG
        num = locserver(host, objstr, &rh, &st);  /* call the locserver to locate server on host */
      if (num == 0) {                        /* if unable to locate server on host print message and exit */
        printf("Failed to communicate with server on %s\n\t%s\n",
         host, dcemsg(st));
        exit(1);
        }
             time(&endexectime);
        endcputime=clock();
printf("Time elapsed to find DCE servers to communicate with cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);


#else
/*********************************Look up in the name
service***************************************************/
        num_entry = findserver(group_name, objstr, C_IFSPEC, &results, MaxSrv,
               &st);  /* All work is done in this routine */

        if (num_entry == 0)  {  /* No servers found registered */
      printf("Lookup for WSM Servers Failed - %s\n", dcemsg(st));
      }

/*
 * Sort out the unique servers in the list of handles
 */
   j = s_handle(num_entry, &nsrv, results, index, proto);

/*
 *  For each server found running, complete the binding information
 */
   for (i=0; i<nsrv; i++)  {
      rpc_mgmt_set_com_timeout(results->binding_h[proto[i]],
          rpc_c_binding_min_timeout, &st); /* timeout quickly */
      rpc_ep_resolve_binding(results->binding_h[proto[i]],
                        C_IFSPEC, &st);
      }  /* End for each server loop */

 time(&endexectime);
endcputime=clock();
printf("Time elapsed to find DCE servers to communicate with cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

       for (i=0; i < nsrv; i++) { /* Output the menu for servers */
        printf("%2d.",i);

        }

        printf("\n\nPick the server to communicate with. Enter the number\n");
        memset(menu,'\0',sizeof(menu));
        getUinput(menu,sizeof(menu));
        rh = results->binding_h[proto[atoi(menu)]];
        printf("Communicating with the web service manager on...\n");
        p_handle(rh);

/************************************************************************
*****************************************/
#endif


                do
            {
                system("clear");
                printf("Enter one of the following\n");
                printf("1. Create a new WSR account.\n");
                printf("2. Create a new WSM account. (Creates a new WSR account too)\n");
                printf("3. Delete a WSM account.\n");
                printf("4. Delete a WSR account.\n");
                        printf("5. List WSM accounts.\n");
                        printf("6. List WSR accounts.\n");
                        printf("7. Reset password for WSM account.\n");
                        printf("8. Reset password for WSR account.\n");
                printf("9. Create WSR accounts for the requests received on the web.\n");
                printf("q. Quit the menu.\n");
                        memset(menu,'\0',sizeof(menu));
                 getUinput(menu,sizeof(menu));
            }while(strlen(menu)==0);


                if(!strncmp(menu,"1",1) || !strncmp(menu,"2",1) || !strncmp(menu,"3",1) ||
!strncmp(menu,"4",1))
            {

                userInfo.name = (char*)malloc(sizeof(char)*64);
                userInfo.word = (char*)malloc(sizeof(char)*64);
                userInfo.address = (char*)malloc(sizeof(char)*256);
                userInfo.phoneNo = (char*)malloc(sizeof(char)*64);
                userInfo.email = (char*)malloc(sizeof(char)*64);

                        if(!strncmp(menu,"1",1))
                                type=1;
                        else
                                type=2;

                docreate = getUserInfo(&userInfo,atoi(menu));
                if(debug)
                {
                    printf("Name *%s*\n",userInfo.name);
                    printf("Word *%s*\n",userInfo.word);
                    printf("Add *%s*\n",userInfo.address);
                    printf("Phone *%s*\n",userInfo.phoneNo);
                    printf("Email *%s*\n",userInfo.email);
                }
                if(!docreate)
                {
                if(!strncmp(menu,"1",1) || !strncmp(menu,"2",1))
                               {

                                       if(!strncmp(menu,"1",1))
                                type=WSR;
                        else
                                type=WSM;
                                       time(&startexectime);
                         startcputime=clock();

                                       createAccount(rh,&userInfo,&type,busid,&status);
                        printf("\n");

                                         time(&endexectime);
                                endcputime=clock();
                                printf("Time elapsed to create an account cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);

                        if(status==0)
                                printf("Account created with bus id *%s*\n",busid);
                        else
                                printf("%s\n",busid);
                }
                                else
                                {
                                       if(!strncmp(menu,"3",1))
                                type=WSM;
                        else
                                type=WSR;
                                       time(&startexectime);
                        startcputime=clock();

                                       deleteAccount(rh,&userInfo,&type,&status);
                        printf("\n");

                                       time(&endexectime);
                         endcputime=clock();
                        printf("Time elapsed to delete an account cpu %ld exec
%ld\n",endcputime-startcputime,endexectime-startexectime);

                        if(status==0)
                          printf("Account Deleted\n");
                        else
                          printf("Error in deleting account\n");
                    }
                }

            free(userInfo.name);
            free(userInfo.word);
            free(userInfo.address);
            free(userInfo.phoneNo);
            free(userInfo.email);
        }


             if(!strncmp(menu,"5",1) || !strncmp(menu,"6",1))
        {


                if(!strncmp(menu,"5",1))
               type=WSM;
            else
               type=WSR;

            userNames.first=0;
            userNames.size=1;
            strcpy(userNames.array,"");

            userIds.first=0;
            userIds.size=1;
            strcpy(userIds.array,"");

                time(&startexectime);
            startcputime=clock();

            getAccounts(rh,&userNames,&userIds,&type,&status);
               printf("\n");

                 time(&endexectime);
             endcputime=clock();
             printf("Time elapsed to get list of accounts cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

                printf("\nThe following accounts were found\n");
               printTwoCol(userNames.array,userIds.array,"User Names","User Ids");
             }

             if(!strncmp(menu,"7",1) || !strncmp(menu,"8",1))
         {

            userInfo.name = (char*)malloc(sizeof(char)*64);
            userInfo.word = (char*)malloc(sizeof(char)*64);
            userInfo.address = (char*)malloc(sizeof(char)*256);
            userInfo.phoneNo = (char*)malloc(sizeof(char)*64);
            userInfo.email = (char*)malloc(sizeof(char)*64);
```

```
                if(!strncmp(menu,"7",1))
                      type=WSM;
                else
                      type=WSR;

                docreate = getUserInfo(&userInfo,atoi(menu));
                if(debug)
                {
                      printf("Name *%s*\n",userInfo.name);
                      printf("Word *%s*\n",userInfo.word);
                      printf("Add *%s*\n",userInfo.address);
                      printf("Phone *%s*\n",userInfo.phoneNo);
                      printf("Email *%s*\n",userInfo.email);
                }
                if(!docreate)
                {

                           time(&startexectime);
                            startcputime=clock();

                                           resetAccount(rh,&userInfo,&type,&status);
                            printf("\n");

                                  time(&endexectime);
                  endcputime=clock();
                  printf("Time elapsed to reset an account cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);
                                           if(status==0)
                                 printf("Account Reset successful\n" );
                            else
                                 printf("Reset Failed\n");
                }

                free(userInfo.name);
                free(userInfo.word);
                free(userInfo.address);
                free(userInfo.phoneNo);
                free(userInfo.email);
          }


                  if(!strncmp(menu,"9",1))
           {

                           userInfo.name = (char*)malloc(sizeof(char)*64);
                userInfo.word = (char*)malloc(sizeof(char)*64);
                userInfo.address = (char*)malloc(sizeof(char)*256);
                userInfo.phoneNo = (char*)malloc(sizeof(char)*64);
                userInfo.email = (char*)malloc(sizeof(char)*64);

                           type=WSR;
```

```
                          if((fp=fopen(REQUESTFILE,"r"))!=NULL)
                         {

                           while(fgets(buffer,256,fp)!=NULL)
                          {

                                        memset(filename,'\0',64);
                                        memset(command,'\0',64);
                                        temp=strtok(buffer,"|");
                          i=1;

                          while((temp=strtok(NULL,"|"))!=NULL)
                          {
                                if(i==1)strcpy(userInfo.name,temp);
                                if(i==3)strcpy(userInfo.word,temp);
                                if(i==5)strcpy(userInfo.address,temp);
                                if(i==7)strcpy(userInfo.email,temp);
                                if(i==9)strcpy(userInfo.phoneNo,temp);
                                i++;
                          }

                                        if(debug)
                               {
                                printf("Name *%s*\n",userInfo.name);
                                printf("Word *%s*\n",userInfo.word);
                                printf("Add *%s*\n",userInfo.address);
                                printf("Phone *%s*\n",userInfo.phoneNo);
                                printf("Email *%s*\n",userInfo.email);
                          }

                                        createAccount(rh,&userInfo,&type,busid,&status);
                          printf("\n");
                          if(status==0)
                                        {
                                                printf("Account created with bus id *%s* for
user %s\n",busid,userInfo.name);
                                                sprintf(filename,".%s",userInfo.name);
                                                if((fp1=fopen(filename,"w"))!=NULL)
                                {
                                fprintf(fp1,"Account has been created with username %s to
access the web service registry.\n",userInfo.name);
                                fprintf(fp1,"The web service registry can be accessed at
http://btest.rnet.missouri.edu/webservices.");
                                                fclose(fp1);
                                                sprintf(command,"mail %s <
%s",userInfo.email,filename);
                                                system(command);
                                                unlink(filename);
                                                }
                                        }
                                        else
                              printf("%s\n",busid);
                                        sleep(3);
                          }

                          }

                   }
}

int getUserInfo(accountInfo *userInfo, int func)
{

   char answer[8];
   int i,value =0;
   char path[128];
      int len;
   char check[64];
      memset(answer,'\0',8);

   do{
            memset(userInfo->name,'\0',64);
            do{
                  printf("Enter the username:\n");
                  getUinput(userInfo->name,64);
                  if(debug)printf("*%s*\n",userInfo->name);
             }while(strlen(userInfo->name)==0);


                   if(func!=3 && func!=4)
                   {
                   memset(userInfo->word,'\0',64);
                   do{
                            strcpy(userInfo->word,getpass("Enter the password:"));
                   if(debug)printf("*%s*\n",userInfo->word);
                   }while(strlen(userInfo->word)==0);
                  code(userInfo->word);
                   }

                   if(func==1 || func==2 )
                   {

                  memset(userInfo->address,'\0',256);
                        printf("Enter the Address (Optional): (Not more than 200 characters)\n");
                        getUinput(userInfo->address,256);
                        if(debug)printf("*%s*\n",userInfo->address);
                  for(i=0;i<strlen(userInfo->address);++i)
                        if(userInfo->address[i]==',')userInfo->address[i]='.';
                  memset(userInfo->phoneNo,'\0',64);
                        printf("Enter the Phone number (Optional):\n");
```

```
                getUinput(userInfo->phoneNo,64);
                if(debug)printf("*%s*\n",userInfo->phoneNo);
            memset(userInfo->email,'\0',64);

                printf("Enter the email address (Optional):\n");
                getUinput(userInfo->email,64);
                if(debug)printf("*%s*\n",userInfo->email);
             }

             if(func==7 || func==8)
             {
             memset(check,'\0',64);
                     do{
            strcpy(check,getpass("Re-Enter the password:"));
            if(debug)printf("*%s*\n",check);
                }while(strlen(check)==0);
            code(check);

                    if (strcmp(check,userInfo->word))
                    {
                            printf("The passwords don't match, re-enter the information\n");
                            continue;
             }
             }

             do{
            printf("If the information entered is correct enter 'y' or enter 'n' to re-enter the
information\n");
            printf("To cancel enter 'c'\n");
            getUinput(answer,sizeof(answer));
         }while(strncmp(answer,"n",1) && strncmp(answer,"c",1) &&
strncmp(answer,"y",1));


        if(debug)printf("*%s*\n",answer);
    }while(strncmp(answer,"y",1) && strncmp(answer,"c",1));

    if(!strncmp(answer,"c",1))
        value=1;
    if(debug)printf("%d",value);
    return value;
}

/************************************************************************
 * Name : code
 * Purpose: To encrypt a word.
 * Function: To use the system call "crypt" to encode the string.
 * Returns:  copies the encoded string in the buffer sent by the calling function.
 ************************************************************************/

void code(char *session_key)
```

```
{
        char *res;
        char session_salt[3];
        session_salt[0]=session_key[strlen(session_key)-1];
        session_salt[1]=session_key[0];
        session_salt[2]='\0';
        res = crypt(session_key, session_salt);
        strcpy(session_key,res);
}

void printTwoCol(char *col1,char* col2,char *title1,char *title2)
{
                char *temp,*temp1,*value,*value1;
                int count=0;

                printf("\n  %-85s %s\n\n",title1,title2);
                                while((temp = strrchr(col1,','))!=NULL)
                                {
                                        printf("%d.",++count);
                                        *temp='\0';
                                        if((temp=strrchr(col1,','))!=NULL)
                                        {
                                                value=temp+1;
                                                printf("%-80s",value);
                                        }
                                        else
                                                printf("%-80s",col1);

                                        temp1 = strrchr(col2,',');
                                        *temp1='\0';
                                        if((temp1=strrchr(col2,','))!=NULL)
                                        {
                                                value1=temp1+1;
                                                printf("\t%s\n",value1);
                                        }
                                        else
                                                printf("\t%s\n",col2);

                                }
}
```

Appendix C: Source Code to Implement Web Service Registry

It contains the source code of the components (programs) used to implement the Web Service Registry.

Web Interface

A) HTML pages

1) Index.htm

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Web Registry</title>
</head>

<frameset rows="64,*">
  <frame name="banner" scrolling="no" noresize target="contents" src="title.htm">
  <frameset cols="239,*">
    <frame name="contents" target="main" src="options.htm">
    <frame name="main" src="home.htm">
  </frameset>
  <noframes>
<body background="grey.bmp">

  <p>This page uses frames, but your browser doesn't support them.</p>

  </body>
  </noframes>
</frameset>

</html>
```

2) Home.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Welcome to the web service registry</title>
</head>
```

```
<body background="grey.bmp">

<p align="left"><font size="4">Web Service Registry</font> is a data repository
of the registered web-services. It is medium to announce the availability of
your web-service to the world. Publish your web-service with this registry and
have it searchable and accessible to the clients. In order to publish a
web-service you will be required to set up an account. Once the account is
created you can publish web-services with the registry. </p>
<p align="left"><font size="4">Create Account</font></p>
<p align="left">The information requested&nbsp; is straight-forward and consists
of a username, password, company's postal address, contact number and email
address. The information is submitted to the web administrator. The
administrator approves the account based on the information provided by the
user. A confirmation email is sent to the user once the account is created.</p>
<p align="left"><font size="4">Update Account</font></p>
<p align="left">Account information can be updated using this option. Fill in
the information for the fields that need to be updated and check the boxes for
the respective fields. If a check box is left unchecked that field will not be
updated.</p>
<p align="left"><font size="4">Publish Service</font></p>
<p align="left">To publish a service you will need to provide the following
information for the web-service so that it is easily searchable by the clients.
Not all the information is mandatory but it will increase the search hits for
the web-service.</p>
<p align="left"><b>Service Name</b>: Name of the service; <b>Service Description</b>:
A brief description of the web-service; <b>Keywords</b>: Words that would best
describe the web-service; <b>URL</b>: URL of the documentation page, if any; <b>
Service Access Points</b>: Urls of wsdl files used to call the web-service; <b>
Functions</b>: Names of the functions implemented by the web-service and their
input and output arguments.</p>
<p align="left">A published web-service will be given a unique id that would be
used to refer the web-service.</p>
<p align="left"><font size="4">Update Service</font></p>
<p align="left">In addition to the information for the fields that need to be
updated, service id of the web-service is required. Check the boxes for the
fields that need to be updated. If the box is not checked, that piece of
information would not be updated. </p>
<p align="left"><font size="4">Delete Service</font></p>
<p align="left">Provides a way to delete a web-service from the registry.
Service id of the web-service is required.</p>
<p align="left"><b><font size="4">Search the Registry</font></b></p>
<p align="left">It will search the registry for the services that match the
criteria. The search is based on: <b>Keywords</b>: Words relevant to the
web-service; <b>Input Parameters</b>: Argument(s) accepted by the function of
the web-service; <b>Output Parameters</b>: Argument(s) returned by the function
of the web-service. Either OR or AND can be used between the search fields based
on the requirement.</p>
<p align="left"><font size="4">Service Information</font></p>
<p align="left">It retrieves information for a given service based on the
service id. </p>
<p align="left"><font size="4">List Services for an account</font></p>
```

```
<p align="left">Provides information about the account along-with the services
published under the account.</p>
<p align="left"><b><font size="4">Web-Service Interface</font></b></p>
<p align="left">The web-service registry can also be accessed programmatically
using the web-services it implements. This link will provide information about
the web-service interfaces for the web registry.&nbsp; There are two
web-services, one is the admin interface and the other is the inquire interface.
</p>
<p align="left"><font size="4">Admin Interface</font>: It&nbsp; provides a
web-service to do the admin operations such as creating an account, publishing a
service, etc from within a program. </p>
<p align="left"><font size="4">Inquire Interface</font>: It provides a
web-service to search the web registry, to inquire about the access points of a
web-service, etc.</p>

</body>

</html>
```

3) Options.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Administrative Options</title>
<base target="main">
</head>

<body background="grey.bmp">

<p><b><font size="4">Administrative Options</font>:</b></p>
<p>&nbsp;&nbsp;&nbsp; <a href="create_account.htm">Create Account</a></p>
<p>&nbsp;&nbsp;&nbsp; <a href="update_account.htm">Update Account</a></p>
<p>&nbsp;&nbsp;&nbsp; <a href="create_service.htm">Publish Service</a></p>
<p>&nbsp;&nbsp;&nbsp; <a href="update_service.htm">Update Service</a></p>
<p>&nbsp;&nbsp;&nbsp; <a href="delete_service.htm">Delete Service</a></p>
<p><font size="4" color="#FF0000"><a href="search_webservice_criteria.htm">Search the
Registry</a></font></p>
<p><a href="search_webservice_serviceid.htm"><font size="4">
Service Information</font></a></p>
<p><font size="4" color="#FF0000"><a href="list_services_account.htm">List
service(s) for an account</a></font></p>
<p><a href="/cgi-bin/getWebRegInfo"><font size="4">Web-Service Interface of the
Registry</font></a></p>

<p><a href="home.htm"><font size="4">Home</font></a></p>
```

```
</body>

</html>
```

4) Create_Account.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Enter Account Information</title>
</head>

<body background="grey.bmp">

<p><b>Fill the following form to create an account. </b></p>

<p><b>The approval of the account is at the discretion of the web administrator
based on the information provided by the user.</b></p>

<p><b>Please provide valid information in the form below.</b></p>

<p><b>The confirmation of the account creation will be sent via email. </b></p>

<p>Enter Account Information:</p>

<form method="POST" action="/cgi-bin/createAccount">

 <p>Username:&nbsp;
 <input type="text" name="username" size="20"></p>
 </p>
<p>Password:&nbsp;
  <input type="password" name="password" size="20"></p>
<p>Address:&nbsp;&nbsp;
  <input type="text" name="address" size="50"></p>
<p>Email:&nbsp;
  <input type="text" name="email" size="20"></p>
<p>Phone Number:&nbsp;
  <input type="text" name="phone" size="20"></p>
<p>
  <input type="submit" value="Submit" name="done"></p>
</form>

</body>

</html>
```

5) Create_Service.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Enter the information for your user account</title>

<script language="javascript">

function addfunctions()
{
 var name, input, output, nameval, inputval, outputval;

 name = document.form1.func_name_all.value;

 if(document.form1.func_name.value)
 nameval=document.form1.func_name.value;
 else nameval="NULL";

 if(name)
 document.form1.func_name_all.value = name+"|"+nameval;
 else document.form1.func_name_all.value = nameval;


 input = document.form1.func_input_all.value;

 if(document.form1.func_input.value)
 inputval=document.form1.func_input.value;
 else inputval="NULL";

 if(input)
 document.form1.func_input_all.value = input+"|"+inputval;
 else document.form1.func_input_all.value = inputval;


 output = document.form1.func_output_all.value;

 if(document.form1.func_output.value)
 outputval=document.form1.func_output.value;
 else outputval="NULL";

 if(output)
 document.form1.func_output_all.value = output+"|"+outputval;
 else document.form1.func_output_all.value = outputval;

 document.form1.func_name.value="";
 document.form1.func_input.value="";
 document.form1.func_output.value="";

}

function addlastfunction()
{
  var name, input, output, nameval, inputval, outputval;

 name = document.form1.func_name_all.value;

 if(document.form1.func_name.value)
 nameval=document.form1.func_name.value;
 else nameval="NULL";

 if(name)
 document.form1.func_name_all.value = name+"|"+nameval;
 else document.form1.func_name_all.value = nameval;


 input = document.form1.func_input_all.value;

 if(document.form1.func_input.value)
 inputval=document.form1.func_input.value;
 else inputval="NULL";

 if(input)
 document.form1.func_input_all.value = input+"|"+inputval;
 else document.form1.func_input_all.value = inputval;


 output = document.form1.func_output_all.value;

 if(document.form1.func_output.value)
 outputval=document.form1.func_output.value;
 else outputval="NULL";

 if(output)
 document.form1.func_output_all.value = output+"|"+outputval;
 else document.form1.func_output_all.value = outputval;


 document.form1.submit();
}
</script>
</head>

<body background="grey.bmp">

<form name="form1" method="POST" action="/cgi-bin/createService"
onsubmit="addlastfunction();">
  <p>
  <b>Fill the following form to publish a service</b></p>
  <p>
  Enter the information for your user account:</p>
  <p>Username:&nbsp;&nbsp;&nbsp; <input type="text" name="username" size="20"></p>
  <p>Password:&nbsp;&nbsp;&nbsp; <input type="password" name="password"
size="20"></p>
  <p>Enter the information for the service to be published:</p>
  <p>Service Name:&nbsp;&nbsp;&nbsp; <input type="text" name="serviceName"
size="20"></p>
  <p>Service Description:&nbsp;&nbsp;&nbsp;
  <input type="text" name="serviceDesc" size="50"></p>
  <p>Keywords (separated by commas):&nbsp;&nbsp;&nbsp; <input type="text"
name="keywords" size="20"></p>
  <p>URL of the documentation page for this service:&nbsp;&nbsp;&nbsp;
  <input type="text" name="docUrl" size="50"></p>
  <p>Service Access Points (separated by commas):&nbsp;&nbsp;&nbsp;
  <textarea rows="2" name="accessPoints" cols="100"></textarea></p>
  <p>Functions of the service</p>
  <p>Name:&nbsp;&nbsp;&nbsp; <input type="text" name="func_name" size="20"></p>
  <p>Input Parameters (separated by commas):&nbsp;&nbsp;&nbsp;
  <input type="text" name="func_input"  size="20"></p>
  <input type="hidden" name="func_name_all">
  <input type="hidden" name="func_input_all">
 <input type="hidden" name="func_output_all">
 <p>Output Parameters (separated by commas):&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="text" name="func_output"  size="20"> </p>
  <p>To add more functions click here&nbsp;&nbsp;&nbsp;
  <input type="button" value="More functions" name="B1" onclick="addfunctions();"> </p>
  <p>When done click submit&nbsp;&nbsp;&nbsp;&nbsp; </p>
  <p><input type="submit" value="Submit" name="done"></p>
</form>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>&nbsp;</p>

</body>

</html>
```

6) Delete_Service.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
</head>
```

```
<body background="grey.bmp">

<p><b>Fill the following form to delete a service.</b></p>

<p>Give the account information:</p>
<form method="POST" action="/cgi-bin/deleteService">
  <p>
  Username:&nbsp;&nbsp;&nbsp; <input type="text" name="username" size="20"></p>
  <p>Password:&nbsp;&nbsp;&nbsp; <input type="password" name="password"
size="20"></p>
  <p>Enter the id of the service to be deleted</p>
  <p>Service Id:&nbsp;&nbsp;&nbsp; <input type="text" name="serviceid" size="20"></p>
  <p><input type="submit" value="Submit" name="done"></p>
</form>

</body>

</html>
```

7) List_Services_account.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
</head>

<body background="grey.bmp">

<p><b>Fill the following form to list services published under an account</b></p>

<p>Give the account information:</p>
<form method="POST" action="/cgi-bin/getServiceByAccount">
  <p>
  Username:&nbsp;&nbsp;&nbsp; <input type="text" name="username" size="20"></p>
  <p>Password:&nbsp;&nbsp;&nbsp; <input type="password" name="password"
size="20"></p>
  <p><input type="submit" value="Submit" name="done"></p>
</form>

</body>

</html>
```

8) Search_webservice_Criteria.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Search a web</title>
</head>

<body background="grey.bmp">

<form method="POST" action="/cgi-bin/searchServices">
  <p>
  <b>Provide the following information (at least one field) to search a web-service</b> </p>
  <p>Enter the keywords (separated by commas):&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="text" name="keywords" size="20"></p>
  <p>Enter the Input Parameters (separated by commas):&nbsp;&nbsp;&nbsp;
  <input type="text" name="input" size="20"></p>
  <p>Enter the Output Parameters (separated by commas):&nbsp;&nbsp;&nbsp;
  <input type="text" name="output" size="20"></p>
  <p><input type="checkbox" name="OR" value="true"> Use &quot;OR&quot; between the
fields</p>
  <p><input type="checkbox" name="AND" value="true"> Use &quot;AND&quot; between the
fields</p>
  <p><input type="submit" value="Submit" name="B1"></p>
</form>
<p>&nbsp;</p>

</body>

</html>
```

9) Search_weservice_serviceid.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Enter the service id for which information is to be retrieved</title>
</head>

<body background="grey.bmp">

<form method="POST" action="/cgi-bin/getServiceById">
  <p>
  Enter the service id for which information is to be retrieved:</p>
  <p><input type="text" name="serviceid" size="32"></p>
  <p><input type="submit" value="Submit" name="done"></p>
</form>

</body>

</html>
```

10) Update_Account.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Password</title>
</head>

<body background="grey.bmp">

<form method="POST" action="/cgi-bin/updateAccount">
  <p><b>Fill the following form to update an account.</b></p>
  <p>Enter Account username and password:</p>
  <p>Username:&nbsp;&nbsp;&nbsp;
  <input type="text" name="username" size="20"></p>
  <p>Current Password:&nbsp;&nbsp;&nbsp;
  <input type="password" name="oldpassword" size="20"></p>
  <p>Check the box for the field that needs to be updated.</p>
  <p><input type="checkbox" name="UpdatePass" value="true">
Password:&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="password" name="password" size="20"></p>
  <p><input type="checkbox" name="UpdateAddress" value="true">
Address:&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="text" name="address" size="50"></p>
  <p><input type="checkbox" name="UpdateEmail" value="true"> Email:&nbsp;&nbsp;&nbsp;
  <input type="text" name="email" size="20"></p>
  <p><input type="checkbox" name="UpdatePhone" value="true"> Phone
Number:&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="text" name="phone" size="20"></p>
  <p><input type="submit" value="Submit" name="done"></p>
</form>

</body>

</html>
```

11) Update_service.htm

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Enter the information for your user account</title>

<script language="javascript">

function addfunctions()
{
 var name, input, output, nameval, inputval, outputval;

 name = document.form1.func_name_all.value;

 if(document.form1.func_name.value)
 nameval=document.form1.func_name.value;
 else nameval="NULL";

 if(name)
 document.form1.func_name_all.value = name+"|"+nameval;
 else document.form1.func_name_all.value = nameval;


 input = document.form1.func_input_all.value;

 if(document.form1.func_input.value)
 inputval=document.form1.func_input.value;
 else inputval="NULL";

 if(input)
 document.form1.func_input_all.value = input+"|"+inputval;
 else document.form1.func_input_all.value = inputval;


 output = document.form1.func_output_all.value;

 if(document.form1.func_output.value)
 outputval=document.form1.func_output.value;
 else outputval="NULL";

 if(output)
 document.form1.func_output_all.value = output+"|"+outputval;
 else document.form1.func_output_all.value = outputval;

 document.form1.func_name.value="";
 document.form1.func_input.value="";
 document.form1.func_output.value="";

}

function addlastfunction()
{
  var name, input, output, nameval, inputval, outputval;

 name = document.form1.func_name_all.value;

 if(document.form1.func_name.value)
 nameval=document.form1.func_name.value;
 else nameval="NULL";

 if(name)
 document.form1.func_name_all.value = name+"|"+nameval;
 else document.form1.func_name_all.value = nameval;


 input = document.form1.func_input_all.value;

 if(document.form1.func_input.value)
 inputval=document.form1.func_input.value;
 else inputval="NULL";

 if(input)
 document.form1.func_input_all.value = input+"|"+inputval;
 else document.form1.func_input_all.value = inputval;


 output = document.form1.func_output_all.value;

 if(document.form1.func_output.value)
 outputval=document.form1.func_output.value;
 else outputval="NULL";

 if(output)
 document.form1.func_output_all.value = output+"|"+outputval;
 else document.form1.func_output_all.value = outputval;


 document.form1.submit();
}
</script>


</head>

<body background="grey.bmp">
```

```
<form name="form1" method="POST" action="/cgi-bin/updateService"
onsubmit="addlastfunction();">
  <p>
  <b>Fill the following form to update a service.</b></p>
  <p>
  Enter the information for your user account:</p>
  <p>Username:&nbsp;&nbsp;&nbsp; <input type="text" name="username" size="20"></p>
  <p>Password:&nbsp;&nbsp;&nbsp; <input type="password" name="password"
size="20"></p>
  <p>Enter the service Id of the service to be updated:</p>
  <p>Service Id:&nbsp;&nbsp;&nbsp; <input type="text" name="serviceid" size="40"></p>
  <p>Check the box for the field that needs to be updated.</p>
  <p><input type="checkbox" name="UpdateName" value="true">&nbsp;&nbsp;&nbsp; Service
  Name:&nbsp;&nbsp; <input type="text" name="serviceName" size="20"></p>
  <p><input type="checkbox" name="UpdateDesc" value="true">&nbsp;&nbsp;&nbsp; Service
Description:&nbsp;&nbsp;&nbsp;
  <input type="text" name="serviceDesc" size="50"></p>
  <p><input type="checkbox" name="UpdateKeywords" value="true">&nbsp;&nbsp;&nbsp;
Keywords (separated by commas):&nbsp;&nbsp;&nbsp; <input type="text" name="keywords"
size="20"></p>
  <p><input type="checkbox" name="UpdateDocUrl" value="true">&nbsp;&nbsp;&nbsp; URL
of the documentation page for this service:&nbsp;&nbsp;&nbsp;
  <input type="text" name="docUrl" size="50"></p>
  <p><input type="checkbox" name="AddAccessPoints" value="true">&nbsp;&nbsp;&nbsp;
Service Access Points (separated by commas)
  to be added:&nbsp;&nbsp;&nbsp;
  <textarea rows="2" name="accessPointsAdd" cols="100"></textarea></p>
  <p><input type="checkbox" name="DeleteAccessPoints" value="true">&nbsp;&nbsp;&nbsp;
Service Access Points (separated by commas)
  to be deleted:&nbsp;&nbsp;&nbsp;
  <textarea rows="2" name="accessPointsDelete" cols="100"></textarea></p>
  <p><input type="checkbox" name="DeleteFunctions" value="true">&nbsp;&nbsp;&nbsp;
Function
  names (separated by commas) to be deleted&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="text" name="functionsDelete" size="20"></p>
  <p><input type="checkbox" name="AddFunctions" value="true">&nbsp;&nbsp;&nbsp;
Functions to
  be added </p>
  <p>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
  Name:&nbsp;&nbsp;&nbsp; <input type="" name="func_name" size="20"></p>
  <p>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
  Input Parameters (separated by commas):&nbsp;&nbsp;&nbsp;
  <input type="text" name="func_input" size="20"></p>
  <p>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
  Output Parameters (separated by commas):&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="text" name="func_output" size="20"></p>
  <input type="hidden" name="func_name_all">
  <input type="hidden" name="func_input_all">
  <input type="hidden" name="func_output_all">
  <p>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; To add
  more functions click here&nbsp;&nbsp;&nbsp;&nbsp;
  <input type="button" value="More functions" name="BI" onclick="addfunctions();"></p>
  <p>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; When
  done click submit</p>
  <p><input type="submit" value="Submit" name="done"></p>
</form>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>&nbsp;</p>

</body>

</html>
```

B) Remote programs (CGIs)

1) CreateAccount.c

```
#include <stdio.h>
#include <stdlib.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char *db_String;
char *read_POST();
int debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

#define REQUESTFILE "/usr/users/khambati/webservices/testexamples/WSM/requestAccount"

void code(char *);

main(){

   char      *query_str;            /* input buffer.                */
   entry     ent;
   char     username[MAX_USER];
   char     password[MAX_PASS];
   char     address[MAX_ADDRESS];
   char     email[MAX_EMAIL];
   char     phone[MAX_PHONE];
   FILE          *fp;

/*************************/
   /* not buffer the stdout: */
   /*************************/
```

```
  setbuf(stdout, NULL);


  memset(username,'\0',MAX_USER);
  memset(password,'\0',MAX_PASS);
  memset(address,'\0',MAX_ADDRESS);
  memset(email,'\0',MAX_EMAIL);
  memset(phone,'\0',MAX_PHONE);


  /**********************/
  /* read the cgi input: */
  /**********************/
  query_str = read_POST();

   printf("Content-type: text/html\n\n");
   printf("<html></pre>");
   printf("<title>Create Account</title>\n<body background=\"/webservices/grey.bmp\">");

             if(debug)
                         printf("Query String %s\n",query_str);

 /****************/
  /* get the session id*/
  /****************/


do
{
  getword(ent.val, query_str, '&');
  plustospace(ent.val);
  unescape_url(ent.val);
  getword(ent.name, ent.val, '=');

  if(!strcmp(ent.name,"username"))
             {
                         if(strlen(ent.val)<MAX_USER)
                                     strcpy(username,ent.val);
                         else  strncpy(username,ent.val,MAX_USER-1);
             }
  else if(!strcmp(ent.name,"password"))
             {
                          if(strlen(ent.val)<MAX_PASS)
                                     strcpy(password,ent.val);
                         else  strncpy(password,ent.val,MAX_PASS-1);

                         code(password);
             }
  else if(!strcmp(ent.name,"address"))
      {
                         if(strlen(ent.val)<MAX_ADDRESS)
```

```
                        strcpy(address,ent.val);
                else  strncpy(address,ent.val,MAX_ADDRESS-1);
            }
  else if(!strcmp(ent.name,"email"))
      {
                if(strlen(ent.val)<MAX_EMAIL)
                        strcpy(email,ent.val);
                else  strncpy(email,ent.val,MAX_EMAIL-1);
            }
  else if(!strcmp(ent.name,"phone"))
            {
                if(strlen(ent.val)<MAX_PHONE)
                     strcpy(phone,ent.val);
                 else  strncpy(phone,ent.val,MAX_PHONE-1);
            }

 }while(strlen(query_str)!=0);

  if(debug)
            printf("Username *%s* Pass *%s* Address *%s*\n Email *%s* Phone
*%s*\n",username,password,address,email,phone);

  if(strlen(username)==0 || strlen(password)==0)
            {
                printf("Username or Password can't be null\n");
                printf("</html>");
                exit(1);
            }

            if((fp=fopen(REQUESTFILE,"a+"))!=NULL)
            {
                fprintf(fp,"Username |%s| Word |%s| Address |%s | Email |%s | Phone |%s
|\n",username,password,address,email,phone);

                fclose(fp);
            }

            printf("A request has been submitted to create an account. A confirmation email will be
sent once the account is created\n");
             printf("</pre></body></html>");

}
```

2) CreateService.c

```
#include <stdio.h>
#include <stdlib.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
  int debug;
void code(char *);

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

  char      *query_str;              /* input buffer.                    */
  char                  *temp;
  entry     ent;
  char      username[MAX_USER];
  char      password[MAX_PASS];
  char      serviceName[MAX_SERVICENAME];
  char      serviceDesc[MAX_SERVICEDESC];
  char      keywords[MAX_KEYWORDS];
  char      docUrl[MAX_DOCURL];
  char      accessPoints[MAX_ACCESSPOINTS];
  char           functions[256];
  char      outputParam[512];
  char           inputParam[512];
  char       busid[64];
  char            serviceid[64];
  char           name[16][64];
  char      input[16][128];
  char      output[16][128];
  char SQLstat[256];
             int i,retcode,status,count=0;
      char resultsend[256];
             char        transactionid[64];
    Message_t msend;
  clock_t   clock(),startcputime,endcputime;
  time_t startexectime,endexectime;

  startcputime=clock();
  time(&startexectime);
  /*************************/
  /* not buffer the stdout: */
```

```
 /**************************/
 setbuf(stdout, NULL);


 memset(username,'\0',MAX_USER);
 memset(password,'\0',MAX_PASS);
 memset(serviceName,'\0',MAX_SERVICENAME);
 memset(serviceDesc,'\0',MAX_SERVICEDESC);
 memset(keywords,'\0',MAX_KEYWORDS);
 memset(docUrl,'\0',MAX_DOCURL);
 memset(accessPoints,'\0',MAX_ACCESSPOINTS);
 memset(serviceid,'\0',64);
 memset(functions,'\0',256);
 memset(inputParam,'\0',512);
 memset(outputParam,'\0',512);
              memset(transactionid,'\0',64);


 /**********************/
 /* read the cgi input: */
 /**********************/
 query_str = read_POST();

  printf("Content-type: text/html\n\n");
  printf("<html><pre>");
   printf("<title>Publish Service</title>\n<body background=\"/webservices/grey.bmp\">");
            if(debug)
            printf("*%s*\n",query_str);

/****************/
 /* get the session id*/
 /****************/

do
{
 getword(ent.val, query_str, '&');
 plustospace(ent.val);
 unescape_url(ent.val);
 getword(ent.name, ent.val, '=');

  if(!strcmp(ent.name,"username"))
     {
            if(strlen(ent.val)<MAX_USER)
                  strcpy(username,ent.val);
            else  strncpy(username,ent.val,MAX_USER-1);
     }
 else if(!strcmp(ent.name,"password"))
     {
             if(strlen(ent.val)<MAX_PASS)
                  strcpy(password,ent.val);
            else  strncpy(password,ent.val,MAX_PASS-1);
            code(password);
            }
else if(!strcmp(ent.name,"serviceName"))
            {
            if(strlen(ent.val)<MAX_SERVICENAME)
                            strcpy(serviceName,ent.val);
                        else  strncpy(serviceName,ent.val,MAX_SERVICENAME-1);
            }
else if(!strcmp(ent.name,"serviceDesc"))
    {
                         if(strlen(ent.val)<MAX_SERVICEDESC)
                            strcpy(serviceDesc,ent.val);
                        else  strncpy(serviceDesc,ent.val,MAX_SERVICEDESC-1);
            }
else if(!strcmp(ent.name,"keywords"))
    {
                        if(strlen(ent.val)<MAX_KEYWORDS)
                            strcpy(keywords,ent.val);
                        else  strncpy(keywords,ent.val,MAX_KEYWORDS-1);
            }
else if(!strcmp(ent.name,"docUrl"))
    {
                        if(strlen(ent.val)<MAX_DOCURL)
                            strcpy(docUrl,ent.val);
                        else  strncpy(docUrl,ent.val,MAX_DOCURL-1);
            }
else if(!strcmp(ent.name,"accessPoints"))
    {
                        if(strlen(ent.val)<MAX_ACCESSPOINTS)
                            strcpy(accessPoints,ent.val);
                        else  strncpy(accessPoints,ent.val,MAX_ACCESSPOINTS-1);
            }
else if(!strcmp(ent.name,"func_name_all"))
    {
                        if(strlen(ent.val)<256)
                            strcpy(functions,ent.val);
                        else  strncpy(functions,ent.val,256-1);
            }
else if(!strcmp(ent.name,"func_input_all"))
    {
                        if(strlen(ent.val)<512)
                            strcpy(inputParam,ent.val);
                        else  strncpy(inputParam,ent.val,512-1);
            }
else if(!strcmp(ent.name,"func_output_all"))
            {
                        if(strlen(ent.val)<512)
                              strcpy(outputParam,ent.val);
                        else  strncpy(outputParam,ent.val,512-1);
            }
```

```
}while(strlen(query_str)!=0);

            if(debug){

                         printf("Username *%s* Pass *%s* Service *%s* Desc
*%s*\n",username,password,serviceName,serviceDesc);
                         printf("DocUrl *%s* AccessPoints *%s* \n functions
*%s*\n",docUrl,accessPoints,functions);
                         printf("Input *%s* Output *%s*\n",inputParam,outputParam);
            }
  if(strlen(username)==0 || strlen(password)==0 || strlen(serviceName)==0)
            {
                         printf("Username or Password or Service Name can't be null\n");
                         printf("</html>");
                         exit(1);
            }

            getuuid(serviceid,serviceName);

             memset(resultsend,'\0',256);
      memset(db_String,'\0',256);

     /*  send Multicast message to get the db_String */
      memset ((void *)&msend, 0, sizeof(Message_t));

      memset(msend.msg,'\0',MaxMsg);
      msend.type=htonl(DBASE_QUERY);
      msend.App=htonl(WSR);
      retcode = sendMulticast(&msend,resultsend);
      if ( retcode ==0)
      {
            strcpy(db_String,resultsend);
      }
      else
      {
            printf("Error in Publishing service. Contact Admininstrator\n");
            printf("</html>");
            exit(1);
      }


            status = authenticateUser(username,password,busid);
            if(debug)
            printf("Authentication status %d\n",status);
      if(status!=0)
      {
            printf("%s\n",busid);
            printf("</html>");
            exit(1);
      }
```

```
            sprintf(transactionid,"%ld%s",startexectime,serviceid);

            memset ((void *)&msend, 0, sizeof(Message_t));
        memset(msend.msg,'\0',MaxMsg);

        msend.type=htonl(UPDATE);
        msend.App=htonl(WSR);
        sprintf(msend.msg,"%s,BEGIN ",transactionid);
        sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s');
",WSR_BUSINESS,busid,serviceid);
        strcat(msend.msg,SQLstat);

        sprintf(SQLstat,"INSERT INTO %s VALUES
('%s','%s','%s','%s','%s');",WSR_SERVICE,serviceid,serviceName,serviceDesc,keywords,docUrl)
;
        strcat(msend.msg,SQLstat);

        if((temp=strtok(accessPoints,","))!=NULL)
            {
                sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s');
",WSR_ACCESSPOINTS,serviceid,temp);
                strcat(msend.msg,SQLstat);
            }
            while((temp=strtok(NULL,","))!=NULL)
            {
                sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s');
",WSR_ACCESSPOINTS,serviceid,temp);
                strcat(msend.msg,SQLstat);
            }


            count=0;
            temp=strtok(functions,"|");
            while(temp!=NULL)
            {
                    strcpy(name[count++],temp);
                    temp=strtok(NULL,"|");
            }

            count=0;
            temp=strtok(inputParam,"|");
        while(temp!=NULL)
        {
                    if(strcmp(temp,"NULL"))
           strcpy(input[count++],temp);
                    else strcpy(input[count++],"");
            temp=strtok(NULL,"|");
        }

            count=0;
        temp=strtok(outputParam,"|");
        while(temp!=NULL)
        {
               if(strcmp(temp,"NULL"))
              strcpy(output[count++],temp);
               else strcpy(output[count++],"");
                          temp=strtok(NULL,"|");
        }


               for(i=0;i<count;++i)
               {

                          if(strcmp(name[i],"NULL"))
                          {

               sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s','%s','%s');
",WSR_FUNCTIONS,serviceid,name[i],input[i],output[i]);
                    strcat(msend.msg,SQLstat);
                          }
               }

        strcat(msend.msg," END;");

               if(debug)
                          printf("Query *%s*\n",msend.msg);

        if(sendMulticast(&msend,resultsend))
        {
              printf("Error in publishing service. Contact administrator\n");
               }

        else
        {
               if(!strcmp(resultsend,"OK"))
               {
                    printf("Service published with the service id %s\n",serviceid);
               }
               else
               {
                    printf("Error in publishing service. Contact administrator\n");
               }
        }
               time(&endexectime);
               endcputime=clock();
               if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);
   printf("</pre></body></html>");
}
```

3) DeleteService.c

```
#include <stdio.h>
#include <stdlib.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
int debug;
void code(char *);

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

  char      *query_str;             /* input buffer.                 */
  entry     ent;
  char      username[MAX_USER];
  char      password[MAX_PASS];
  char      busid[64];
  char                  serviceid[64];
  char        SQLstat[256];
  int retcode,st,status;
      char resultsend[256];
      Message_t msend;
 int i,noColumns;
  char        **columns[1];
  char      transactionid[64];
  clock_t clock(),startcputime,endcputime;
  time_t  startexectime,endexectime;
  /**************************/
  /* not buffer the stdout: */
  /**************************/
  setbuf(stdout, NULL);

  startcputime=clock();
  time(&startexectime);

  memset(username,'\0',MAX_USER);
  memset(password,'\0',MAX_PASS);
  memset(serviceid,'\0',64);
             memset(transactionid,'\0',64);

  /**********************/
  /* read the cgi input: */
  /***********************/
  query_str = read_POST();

   printf("Content-type: text/html\n\n");
   printf("<html><pre>");
   printf("<title>Delete Service</title>\n<body background=\"/webservices/grey.bmp\">");

             if(debug)
                         printf("Query String %s\n",query_str);

 /*****************/
  /* get the session id*/
  /*****************/


 do
 {
  getword(ent.val, query_str, '&');
  plustospace(ent.val);
  unescape_url(ent.val);
  getword(ent.name, ent.val, '=');

  if(!strcmp(ent.name,"username"))
             {
                         if(strlen(ent.val)<MAX_USER)
                                     strcpy(username,ent.val);
                         else  strncpy(username,ent.val,MAX_USER-1);
             }
  else if(!strcmp(ent.name,"password"))
             {
                          if(strlen(ent.val)<MAX_PASS)
                                     strcpy(password,ent.val);
                         else  strncpy(password,ent.val,MAX_PASS-1);
                         code(password);
             }
  else if(!strcmp(ent.name,"serviceid"))
      {
                         if(strlen(ent.val)<64)
                                     strcpy(serviceid,ent.val);
                         else  strncpy(serviceid,ent.val,64-1);
             }

 }while(strlen(query_str)!=0);

  if(debug)
             printf("Username *%s* Pass *%s* Serviceid *%s*\n",username,password,serviceid);

  if(strlen(username)==0 || strlen(password)==0 || strlen(serviceid)==0)
             {
                         printf("Username or Password or Serviceid can't be null\n");
                         printf("</html>");
```

```
                exit(1);
        }

        memset(resultsend,'\0',256);
    memset(db_String,'\0',256);

   /*  send Multicast message to get the db_String */
    memset ((void *)&msend, 0, sizeof(Message_t));

    memset(msend.msg,'\0',MaxMsg);
    msend.type=htonl(DBASE_QUERY);
    msend.App=htonl(WSR);
    retcode = sendMulticast(&msend,resultsend);
    if ( retcode ==0)
    {
        strcpy(db_String,resultsend);
    }
    else
    {
        printf("Error in Updating Service. Contact Admininstrator\n");
        printf("</html>");
        exit(1);
    }

    status = authenticateUser(username,password,busid);
    if(debug)
        printf("Authentication status %d\n",status);
    if(status!=0)
    {
        printf("%s\n",busid);
        printf("</html>");
        exit(1);
    }


        sprintf(SQLstat,"SELECT serviceid FROM %s WHERE %s = '%s' AND %s
='%s'",WSR_BUSINESS,WSR_BUSINESS_BUSID,busid,WSR_BUSINESS_SERVICEID,serv
iceid);
    noColumns=1;
    for(i=0;i<noColumns;++i)
    {
        columns[i]=(char **)malloc(sizeof(char**));

    }

    st = fetch(SQLstat,columns,noColumns);

    if(st==0 || st==-1)
    {
        if(st==0)
            printf("Service %s not found for the given user %s",serviceid,username);
            else
                  printf("Error in updating service. Contact Administrator\n");

            for(i=0;i<noColumns;++i)
                  {
                        free(*(columns[i]));
                  }
            exit(1);
      }

      for(i=0;i<noColumns;++i)
                  {
                        free(*(columns[i]));
                  }

             sprintf(transactionid,"%ld%s",startexectime,busid);

   memset ((void *)&msend, 0, sizeof(Message_t));
      memset(msend.msg,'\0',MaxMsg);

      /************************Multicast the given delete******************/
      msend.type=htonl(UPDATE);
      msend.App=htonl(WSR);

      sprintf(msend.msg,"%s,DELETE FROM %s where %s =
'%s'",transactionid,WSR_BUSINESS,WSR_BUSINESS_SERVICEID,serviceid);

             if(debug)
             printf("Query *%s*\n",msend.msg);
     if(sendMulticast(&msend,resultsend))
      {
           printf("Error in deleting Service. Contact administrator\n");
            }

      else
      {
            if(!strcmp(resultsend,"OK"))
            {
                  printf("Service %s deleted\n",serviceid);
            }
            else
            {
                  printf("Error in deleting service. Contact administrator\n");
            }
      }
             time(&endexectime);
             endcputime=clock();
      if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

   printf("</pre></body></html>");
```

```
}
```

4) GetServiceByAccount.c

```
#include <stdio.h>
#include <stdlib.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
void printTwoCol(char *col1,char* col2,char *title1,char *title2);
int debug;

void code(char *);

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

   char       *query_str;               /* input buffer.                    */
   entry      ent;
   char       busid[64];
   char                    username[MAX_USER];
   char                    password[MAX_PASS];
   int retcode;
       char resultsend[256];
       Message_t msend;
  int st,i,noColumns,status;
   char         **columns[4];
   char                    query[256];
   char                    *temp;
   clock_t    clock(),startcputime,endcputime;
   time_t startexectime,endexectime;
/**************************/
   /* not buffer the stdout: */
   /**************************/
   setbuf(stdout, NULL);

   startcputime=clock();
   time(&startexectime);

   memset(busid,'\0',64);


   /**********************/
  /* read the cgi input: */
  /***********************/
  query_str = read_POST();

   printf("Content-type: text/html\n\n");
   printf("<html><pre>");
    printf("<title>Services</title>\n");
   printf("<head>\n");
       printf("<SCRIPT language=javascript>");
       printf("var currentWin=null,currentWin1=null;");

       printf("function OpenWindow(url,dx,dy,WindowName) {\n");
       printf("if (currentWin) currentWin1 = currentWin;");
       printf("currentWin =
window.open(url,WindowName,\"scrollbars=yes,status=no,toolbar=no,menubar=no,location=no,
resizable=yes\");");
       printf("}");
       printf("</script></head><body background=\"/webservices/grey.bmp\">");

   if(debug)
            printf("Query String *%s*\n",query_str);

 /*****************/
  /* get the session id*/
  /*****************/

 do
 {
  getword(ent.val, query_str, '&');
  plustospace(ent.val);
  unescape_url(ent.val);
  getword(ent.name, ent.val, '=');

 if(!strcmp(ent.name,"username"))
      {
            if(strlen(ent.val)<MAX_USER)
                  strcpy(username,ent.val);
            else  strncpy(username,ent.val,MAX_USER-1);
      }
  else if(!strcmp(ent.name,"password"))
      {
             if(strlen(ent.val)<MAX_PASS)
                  strcpy(password,ent.val);
            else  strncpy(password,ent.val,MAX_PASS-1);
                        code(password);
      }


 }while(strlen(query_str)!=0);
```

```
  if(debug)
            printf("USernmae *%s* Pass *%s* \n",username,password);
  if(strlen(username)==0 || strlen(password)==0)
            {
                        printf("Username or Password can't be null\n");
                        printf("</html>");
                        exit(1);
            }


            memset(resultsend,'\0',256);
      memset(db_String,'\0',256);

     /*  send Multicast message to get the db_String */
      memset ((void *)&msend, 0, sizeof(Message_t));

      memset(msend.msg,'\0',MaxMsg);
      msend.type=htonl(DBASE_QUERY);
      msend.App=htonl(WSR);
      retcode = sendMulticast(&msend,resultsend);
      if ( retcode ==0)
      {
            strcpy(db_String,resultsend);
      }
      else
      {
            printf("Error in retrieving information. Contact Admininstrator\n");
            printf("</html>");
            exit(1);
      }

            /* Form the query to get the account info */
      sprintf(query,"SELECT %s,%s,%s,%s FROM %s WHERE %s ='%s' AND
%s='%s'",WSR_AUTH_ADD,WSR_AUTH_EMAIL,WSR_AUTH_PHONE,WSR_AUTH_BUS
ID,WSR_AUTH,WSR_AUTH_NAME,username,WSR_AUTH_WORD,password);

      if(debug)printf("In getservicebyAccount with query *%s*\n",query);

      noColumns=4;
         for(i=0;i<noColumns;++i)
      {
            columns[i]=(char **)malloc(sizeof(char**));

      }


      /* call the function to execute the query and return the results */
      st = fetch(query,columns,noColumns);
      if(st==0 || st==-1)
      {
                        if(st==0)
```

```
                 printf("ERROR:Not Authenticated, Check Username and password\n");
                else
                                      printf("ERROR:Contact Administrator\n");
                          printf("</html>");
                         for(i=0;i<noColumns;++i)
                {
                       free(*(columns[i]));
                }

               exit(1);
        }


                 strcpy(busid,strtok(*(columns[3]),","));

                  printf("The information for the given account is:\n");
          printf("Address: %s\n",strtok(*(columns[0]),","));
          printf("Email: %s\n",strtok(*(columns[1]),","));
          printf("Phone number: %s\n\n",strtok(*(columns[2]),","));

                 for(i=0;i<noColumns;++i)
                 {
                       free(*(columns[i]));
                 }


                 sprintf(query,"SELECT M.%s,M.%s FROM %s S, %s M WHERE S.%s ='%s' AND
S.%s=
M.%s",WSR_SERVICE_NAME,WSR_SERVICE_SERVICEID,WSR_BUSINESS,WSR_SERV
ICE,WSR_BUSINESS_BUSID,busid,WSR_BUSINESS_SERVICEID,WSR_SERVICE_SERVI
CEID);
                 if(debug)
                         printf("Query *%s*\n",query);

        noColumns=2;
        for(i=0;i<noColumns;++i)
        {
                columns[i]=(char **)malloc(sizeof(char**));

        }

        st = fetch(query,columns,noColumns);

                 if(st==0 || st==-1)
        {
                         if(st==0)
                printf("No service(s) published for the given account\n");
                         else
                printf("Error in Finding service(s) for the given account. Contact Administrator\n");

                for(i=0;i<noColumns;++i)
```

```
            {
                  free(*(columns[i]));
            }
                        printf("</html>");
                        exit(1);
      }

            printf("The following services were found for the given account\n");

            printTwoCol(*(columns[0]),*(columns[1]),"Service Names","Service Ids");

            for(i=0;i<noColumns;++i)
                  {
                        free(*(columns[i]));
                  }
            time(&endexectime);
            endcputime=clock();
      if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

  printf("</pre></body></html>");
}

void printTwoCol(char *col1,char* col2,char *title1,char *title2)
{
            char *temp,*temp1,*value,*value1;
            int count=0,i;
            printf("\n  %-40s  %s\n\n",title1,title2);
                              while((temp = strrchr(col1,','))!=NULL)
                              {
                                    printf("%d.",++count);
                                    *temp='\0';
                                    if((temp=strrchr(col1,','))!=NULL)
                                    {
                                          value=temp+1;
                                    }
                                    else
                                                                              value=col1;
                                   temp1 = strrchr(col2,',');
                                    *temp1='\0';
                                    if((temp1=strrchr(col2,','))!=NULL)
                                    {
                                          value1=temp1+1;
                                    }
                                    else
                                                                              value1=col2;
printf("<a href=javascript:OpenWindow(\"/cgi-
bin/getServiceById?serviceid=%s\",20,0,\"%d\")>%s</a>",value1,count,value);
                                                for(i=0;i<(abs(40-strlen(value)));++i)printf(" ");
                                          printf("%s\n",value1);
                              }
```

```
}

5) GetServiceById.c

#include <stdio.h>
#include <stdlib.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
int   debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

  char       *query_str;              /* input buffer.                    */
  entry      ent;
  char       serviceid[64];
  int retcode;
      char resultsend[256];
      Message_t msend;
 int st,i,noColumns;
  char         **columns[3];
  char                       query[256];
  char                       *temp;
  clock_t  clock(),startcputime,endcputime;
  time_t startexectime,endexectime;

 /**************************/
  /* not buffer the stdout: */
  /**************************/
  setbuf(stdout, NULL);

               startcputime=clock();
  time(&startexectime);

  memset(serviceid,'\0',64);


  /**********************/
  /* read the cgi input: */
  /**********************/

if(!strcmp(getenv("REQUEST_METHOD"),"POST"))
```

```
                query_str = read_POST();
else query_str=getenv("QUERY_STRING");

    printf("Content-type: text/html\n\n");
    printf("<html><pre>");
     printf("<title>Service Information</title>\n<body background=\"/webservices/grey.bmp\">");
   if(debug)
                printf("Query String *%s*\n",query_str);

  /*****************/
   /* get the session id*/
   /*****************/

  do
  {
   getword(ent.val, query_str, '&');
   plustospace(ent.val);
   unescape_url(ent.val);
   getword(ent.name, ent.val, '=');

  if(!strcmp(ent.name,"serviceid"))
        {
                if(strlen(ent.val)<64)
                        strcpy(serviceid,ent.val);
                else  strncpy(serviceid,ent.val,64-1);
        }

  }while(strlen(query_str)!=0);


   if(debug)
                printf("Serviceid *%s*\n",serviceid);
   if(strlen(serviceid)==0)
                {
                                printf("Service Id can't be null\n");
                                printf("</html>");
                                exit(1);
                }


                memset(resultsend,'\0',256);
        memset(db_String,'\0',256);

       /*  send Multicast message to get the db_String */
        memset ((void *)&msend, 0, sizeof(Message_t));

        memset(msend.msg,'\0',MaxMsg);
        msend.type=htonl(DBASE_QUERY);
        msend.App=htonl(WSR);
        retcode = sendMulticast(&msend,resultsend);
        if ( retcode ==0)
        {
                strcpy(db_String,resultsend);
        }
        else
        {
                printf("Error in Retreiving Information. Contact Admininstrator\n");
                printf("</html>");
                exit(1);
        }

         memset ((void *)&msend, 0, sizeof(Message_t));
        memset(msend.msg,'\0',MaxMsg);

         sprintf(query,"SELECT %s,%s,%s FROM %s  where %s
='%s'",WSR_SERVICE_NAME,WSR_SERVICE_DESCRP,WSR_SERVICE_DOCURL,WSR_
SERVICE,WSR_SERVICE_SERVICEID,serviceid);


         if(debug)
                         printf("Query *%s*\n",query);

        noColumns=3;
        for(i=0;i<noColumns;++i)
        {
                columns[i]=(char **)malloc(sizeof(char**));

        }

        st = fetch(query,columns,noColumns);

         if(st==0 || st==-1)
        {
                printf("Error in Finding service");

                for(i=0;i<noColumns;++i)
                {
                        free(*(columns[i]));
                }
                         printf("</html>");
                         exit(1);
        }

         printf("<h3>Service Information:\n\n</h3>");
         printf("<h5>ServiceId: %s\n</h5>",serviceid);
         if((temp=strtok(*(columns[0]),","))!=NULL)
         printf("<h5>Name: %s\n</h5>",temp);

         if((temp=strtok(*(columns[1]),","))!=NULL)
        printf("<h5>Description: %s\n</h5>",temp);

         if((temp=strtok(*(columns[2]),","))!=NULL)
        printf("<h5>Url of the documentation page: %s\n\n</h5>",temp);

        for(i=0;i<noColumns;++i)
                {
                        free(*(columns[i]));
                }

                /*get the accesspoints for the service */

        sprintf(query,"SELECT %s FROM %s where
%s='%s'",WSR_ACCESSPOINTS_URL,WSR_ACCESSPOINTS,WSR_ACCESSPOINTS_SER
VICEID,serviceid);

        if(debug)
                        printf("Query *%s*\n",query);
                noColumns=1;
        for(i=0;i<noColumns;++i)
        {
               columns[i]=(char **)malloc(sizeof(char**));

        }

        st = fetch(query,columns,noColumns);


        if(st==0 || st==-1)
        {
               printf("<h4>No Accesspoints available for this service right now\n\n</h4>");

                        for(i=0;i<noColumns;++i)
               {
                       free(*(columns[i]));
               }

        }
                else
                {
                        printf("<h4>Access Points for this service are as:\n\n</h4>");
                        if((temp=strtok(*(columns[0]),","))!=NULL)
                        printf("<h5>%s\n</h5>",temp);

                        while((temp=strtok(NULL,","))!=NULL)
                {
                        printf("<h5>%s\n</h5>",temp);
                        }

                        printf("\n");
                        for(i=0;i<noColumns;++i)
                {
                       free(*(columns[i]));
                }
```

```
          }
          /* Getting info about the owner of the service */
     sprintf(query,"select DISTINCT A.%s, A.%s, A.%s from %s A, %s B where B.%s ='%s'
AND B.%s =
A.%s",WSR_AUTH_ADD,WSR_AUTH_EMAIL,WSR_AUTH_PHONE,WSR_AUTH,WSR_B
USINESS,WSR_BUSINESS_SERVICEID,serviceid,WSR_BUSINESS_BUSID,WSR_AUTH_
BUSID);


          noColumns=3;
     for(i=0;i<noColumns;++i)
     {
          columns[i]=(char **)malloc(sizeof(char**));

     }

          if(debug)
          printf("Query *%s*\n",query);

     st = fetch(query,columns,noColumns);

          if(debug)printf("st %d\n",st);

     if(st==0 || st==-1)
     {
          printf("<h4>Information about the publisher not available\n</h4>");

          for(i=0;i<noColumns;++i)
          {
               free(*(columns[i]));
          }

     }
          else
          {
                    printf("<h4>Information about the publisher of the service:\n\n</h4>");
          if((temp=strtok(*(columns[0]),","))!=NULL)
                    printf("<h5>Address: %s\n</h5>",temp);

          if((temp=strtok(*(columns[1]),","))!=NULL)
                    printf("<h5>Email Address: %s\n</h5>",temp);

          if((temp=strtok(*(columns[2]),","))!=NULL)
                    printf("<h5>Phone Number: %s\n</h5>",temp);


                    for(i=0;i<noColumns;++i)
          {
               free(*(columns[i]));
          }
          }
```

```
                time(&endexectime);
                endcputime=clock();
        if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

  printf("</pre></body></html>");
}
```

6) GetWebRegInfo.c

```
#include <stdio.h>
#include <stdlib.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
int   debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

   char       *query_str;              /* input buffer.                  */
   entry      ent;
   char       serviceid[64];
   int retcode;
        char resultsend[256];
        Message_t msend;
  int st,i,noColumns;
   char         **columns[3];
   char                      query[256];
   char                      *temp;
   clock_t clock(),startcputime,endcputime;
   time_t startexectime,endexectime;
 /**************************/
   /* not buffer the stdout: */
   /**************************/
   setbuf(stdout, NULL);

                 startcputime=clock();
   time(&startexectime);

   memset(serviceid,'\0',64);
```

```
    printf("Content-type: text/html\n\n");
    printf("<html><pre>");
     printf("<title>Web Service Registry Information</title>\n");
    printf("<head>\n");
        printf("<SCRIPT language=javascript>");
        printf("var currentWin=null,currentWin1=null;");

        printf("function OpenWindow(url,dx,dy,WindowName) {\n");
        printf("if (currentWin) currentWin1 = currentWin;");
        printf("currentWin =
window.open(url,WindowName,\"scrollbars=yes,status=no,toolbar=no,menubar=no,location=no,
resizable=yes\");");
        printf("}");
        printf("</script></head>\n<body background=\"/webservices/grey.bmp\">");

                memset(resultsend,'\0',256);
        memset(db_String,'\0',256);

       /*  send Multicast message to get the db_String */
        memset ((void *)&msend, 0, sizeof(Message_t));

        memset(msend.msg,'\0',MaxMsg);
        msend.type=htonl(DBASE_QUERY);
        msend.App=htonl(WSR);
        retcode = sendMulticast(&msend,resultsend);
        if ( retcode ==0)
        {
              strcpy(db_String,resultsend);
        }
        else
        {
              printf("Error in Retreiving Information. Contact Admininstrator\n");
              printf("</html>");
              exit(1);
        }

                /* print the link for the admin interface */
                sprintf(query,"SELECT %s FROM %s  where %s
='%s'",WSM_SERVICE_SERVICEID,WSM_SERVICE,WSM_SERVICE_NAME,WSRADMI
N_WEBSERVICE_NAME);

                if(debug)
                                printf("Query *%s*\n",query);

        noColumns=1;
        for(i=0;i<noColumns;++i)
        {
              columns[i]=(char **)malloc(sizeof(char**));

        }
        st = fetch(query,columns,noColumns);

            if(st==0 || st==-1)
        {
            printf("Error in Finding info for the web registry admin web-service");

            for(i=0;i<noColumns;++i)
            {
                free(*(columns[i]));
            }
        }
            else
            {
                    if((temp=strtok(*(columns[0]),","))!=NULL)
                            printf("<a href=javascript:OpenWindow(\"/cgi-
bin/getServiceById?serviceid=%s\",20,0,\"Admin\")>Admin Interface</a>\n",temp);
                    for(i=0;i<noColumns;++i)
            {
                    free(*(columns[i]));
            }

            }
            /* print the link for the inquire interface */
        sprintf(query,"SELECT %s FROM %s  where %s
='%s'",WSM_SERVICE_SERVICEID,WSM_SERVICE,WSM_SERVICE_NAME,WSRINQUI
RE_WEBSERVICE_NAME);

        if(debug)
            printf("Query *%s*\n",query);

        noColumns=1;
        for(i=0;i<noColumns;++i)
        {
            columns[i]=(char **)malloc(sizeof(char**));

        }

        st = fetch(query,columns,noColumns);

        if(st==0 || st==-1)
        {
            printf("Error in Finding info for the web registry Inquire web-service");

            for(i=0;i<noColumns;++i)
            {
                free(*(columns[i]));
            }
            printf("</html>");
            exit(1);
        }
```

```
                else
        {
               if((temp=strtok(*(columns[0]),","))!=NULL)
                       printf("<a href=javascript:OpenWindow(\"/cgi-
bin/getServiceById?serviceid=%s\",20,0,\"Inquire\")>Inquire Interface</a>",temp);
               for(i=0;i<noColumns;++i)
               {
                       free(*(columns[i]));
               }

        }
                time(&endexectime);
                endcputime=clock();
        if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

  printf("</pre></body></html>");
}
```

7) GetWSRAddress.c

```
#include <stdio.h>
#include "mcast.h"
#include <errno.h>
int sendMulticast(Message_t *msend, char *result);

int debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif


main()
{

                Message_t msend;
                int           retcode;
                char          result[1024];
                int            flag=0;
                char          *temp;
                clock_t clock(),startcputime,endcputime;


                 startcputime=clock();
   time(&startexectime);

                memset ((void *)&msend, 0, sizeof(Message_t));
```

```
        memset(msend.msg,'\0',MaxMsg);
                memset(result,'\0',1052);
        msend.type=ntohl(WSR_QUERY); /* Set the type of the multicast message */
        msend.App=htonl(WSR);  /* Which application */

                retcode = sendMulticast(&msend,result);
                if ( retcode ==0)
        {
                                if(strncmp(result,"ERROR",5))
                                                flag=1;
                }

                if(flag)
                {
                                temp=strtok(result,",");
                                while(temp!=NULL)
                                {
                                                printf("%s\n",temp);
                                                temp=strtok(NULL,",");
                                }
                }
                else
                {
                                printf("Can't get the address(s) from the registry\n");
                                printf("Try using the following default addresses\n");
                                printf("http://btest.rnet.missouri.edu/webservices\n");
                }
                time(&endexectime);
                endcputime=clock();
        if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

}



int sendMulticast(Message_t *msend, /* Message structure sent by the calling function to be
multicasted */
                 char *result)    /* Buffer sent by the calling function to copy the result */
{

  const unsigned char loop = 0;
  char c;
  int mport,i;
  unsigned char ttl = 64;
  Message_t mrecv;
  int datalen, n, msglen, addrlen;
  const int on = 1;
  Control_t Control;
  int select;
  int response = 0;
  int retry = RETRIES;
  struct sockaddr_in raddr; /* mcast address we are receiving on */
  struct sigaction act,old;
  struct timeval rcvoldto;
  char  responses[16][1024];
  strcpy(result,"ERROR");
/* Init control structure */
  memset ((void *)&Control, 0, sizeof(Control));

  bzero(&Control,sizeof(Control_t));

       if(debug)printf("In sendMulticast with message *%s* and type \n",msend-
>msg,ntohl(msend->type));

/* Determine who we are */
  if (gethostname((char *)Control.myhost, 128) == -1) {
    perror ("Error getting local host name!\n");
    return 1;
  }
  strncpy ((char *)Control.mgroup, DefaultGroup, 31);

    mport = DefaultPort;

/* First open sockets */
  if ((Control.sfd=socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("Error opening sending socket");
return 1;
  }

if (strlen(DEFAULTINTERFACE)!=0) {
    if (strlen(DEFAULTINTERFACE) > 6) { /* must be an ip address */
        printf("\nUsing the interface with address %s\n\n", DEFAULTINTERFACE);
      Control.iface.sin_addr.s_addr = inet_addr(DEFAULTINTERFACE);
    } else {
       printf("Only valid IP addresses are accepted here!\n");
      close(Control.sfd);
      return 1;
    }

    /* OK, now set the sending interface for this mcast group! */
    if (setsockopt(Control.sfd, IPPROTO_IP,
               IP_MULTICAST_IF, &Control.iface.sin_addr,
               sizeof(Control.maddr.sin_addr)) < 0) {
      perror("setsockopt: IP_MULTICAST_IF");
      close(Control.sfd);
      return 1;
    }

  } else {
    Control.iface.sin_addr.s_addr = INADDR_ANY;
  }
```

```
/* Take care of the sending side ... */

  if (setsockopt(Control.sfd, IPPROTO_IP,
               IP_MULTICAST_TTL, &ttl, sizeof(ttl)) < 0) {
    perror("setsockopt with ttl");
    close(Control.sfd);
    return 1;
  }

  if (setsockopt(Control.sfd, IPPROTO_IP,
               IP_MULTICAST_LOOP, &loop, sizeof(loop)) < 0) {
    perror("setsockopt IP_MULTICAST_LOOP");
    return 1;
  }
/* And the receiving side */
  Control.maddr.sin_family = AF_INET;
  Control.maddr.sin_port = htons(mport);
  Control.maddr.sin_addr.s_addr = inet_addr(Control.mgroup);
  /* set REUSEADDR so that we can have multiple listeners.  */
  if (setsockopt(Control.sfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0) {
    perror("setsockopt with SO_REUSEADDR");
    close(Control.sfd);
    return 1;
  }

 /* set the time span */
 rcvoldto.tv_sec = 8; /* seconds */
 rcvoldto.tv_usec = 0; /* milli seconds */

 /* set RCVTIMEO to wait for the reponses for a specified time span*/
  if (setsockopt(Control.sfd, SOL_SOCKET, SO_RCVTIMEO, &rcvoldto, sizeof(struct timeval))
< 0) {
    perror("setsockopt with SO_RCVTIMEO");
    close(Control.sfd);
    return 1;
  }

/* Sending the message on the multicast address */

/* Clear the message areas and init the msend structure */
  memset ((void *)&mrecv, 0, sizeof(Message_t));

  time_t t = time(0);

      msend->msglen = htonl((msglen = strlen(msend->msg)+1));
      datalen = AppHdrLen + msglen;
         addrlen = sizeof(struct sockaddr_in);

       /*Loop till a response is received or till number of retries */
      while(!response && retry)
      {
            if(debug)printf("Sending message \n");
            if ((n = sendto(Control.sfd, (const void *)msend, datalen, 0, (struct sockaddr
*)&Control.maddr,
                addrlen)) < 0) {
 perror("sendto failed!");
            return 1;
            }

            while(1){    /* Listening for the responses till timer expires */
                if ((n = recv(Control.sfd, (void *)&mrecv, sizeof(Message_t), 0/*, (struct sockaddr
*) &raddr,
                 &addrlen*/)) ==-1) {
                     if (errno == EAGAIN || errno== EWOULDBLOCK){
                          if(debug)printf("Timer expired\n");
                          break;     /* if the timer expires break from the while loop */
                     } else printf("Unable to receive response\n");
                }
                else {
                response++; /* Atleast one response was received */
                memset(responses[response-1],'\0',1024);
                strcpy(responses[response-1],mrecv.msg);
                }
            }
            retry--;
       }

       if(!response){
            if(debug)printf("No response received from any members of the multicast group\n");
            return 1;
       }
       else
       {
            if(debug)printf("%d responses received\n",response);

            /* If the message type is to get a database server's address*/
            if(msend->type==htonl(WSR_QUERY))
            {
                      for(i=0;i<response;++i) /* Find the first available server */
                      {
                           if(debug)printf("repsonse %d *%s* received\n",i+1,responses[i]);
                           strcpy(result,responses[i]);
                           if(strncmp(responses[i],"ERROR",5))
                           break;
                      }
            }

       }

       if(debug)printf("Exiting sendMulticast result *%s*\n",result);

/* Close the socket */
```

```
 close(Control.sfd);

 return 0;
}
```

8) SearchServices.c

```
#include <stdio.h>
#include <stdlib.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
void printTwoCol(char *col1,char* col2,char *title1,char *title2);
int debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

  char       *query_str;              /* input buffer.                  */
  entry     ent;
  char       keywords[MAX_KEYWORDS];
  char       inputParam[256];
  char       outputParam[256];
  char             KeywordsSearchString[512];
  char InputPSearchString[512];
 char OutputPSearchString[512];
 char SQLstat[256];
      char query[1024];
  int retcode;
      char resultsend[256];
      Message_t msend;
 int st,i,noColumns,searchflag=0,ANDflag=0,ORflag=0;
  char         **columns[3];
  char                      *temp;
  clock_t clock(),startcputime,endcputime;
  time_t  startexectime,endexectime;
 /*************************/
  /* not buffer the stdout: */
  /*************************/
  setbuf(stdout, NULL);

  startcputime=clock();
  time(&startexectime);
  memset(keywords,'\0',MAX_KEYWORDS);
  memset(inputParam,'\0',256);
  memset(outputParam,'\0',256);
  memset(InputPSearchString,'\0',512);
  memset(OutputPSearchString,'\0',512);
  memset(KeywordsSearchString,'\0',512);

  /**********************/
  /* read the cgi input: */
  /**********************/
  query_str = read_POST();

   printf("Content-type: text/html\n\n");
   printf("<html><pre>");
    printf("<title>Services Found</title>\n");
  printf("<head>\n");
       printf("<SCRIPT language=javascript>");
       printf("var currentWin=null,currentWin1=null;");

       printf("function OpenWindow(url,dx,dy,WindowName) {\n");
       printf("if (currentWin) currentWin1 = currentWin;");
       printf("currentWin =
window.open(url,WindowName,\"scrollbars=yes,status=no,toolbar=no,menubar=no,location=no,
resizable=yes\");");
       printf("}");
       printf("</script></head>\n<body background=\"/webservices/grey.bmp\">");


              if(debug)
                         printf("Query String *%s*\n",query_str);

 /****************/
  /* get the session id*/
  /****************/

 do
 {
  getword(ent.val, query_str, '&');
  plustospace(ent.val);
  unescape_url(ent.val);
  getword(ent.name, ent.val, '=');

  if(!strcmp(ent.name,"keywords"))
       {
              if(strlen(ent.val)<MAX_KEYWORDS)
                     strcpy(keywords,ent.val);
              else  strncpy(keywords,ent.val,MAX_KEYWORDS-1);
       }
  else if(!strcmp(ent.name,"input"))
       {
            if(strlen(ent.val)<256)
                  strcpy(inputParam,ent.val);
            else  strncpy(inputParam,ent.val,256-1);
       }
  else if(!strcmp(ent.name,"output"))
       {
            if(strlen(ent.val)<256)
                 strcpy(outputParam,ent.val);
            else  strncpy(outputParam,ent.val,256-1);
       }

 if(!strcmp(ent.name,"AND"))
  {
     if(!strcmp(ent.val,"true")) ANDflag=1;
  }
  else if(!strcmp(ent.name,"OR"))
  {
     if(!strcmp(ent.val,"true")) ORflag=1;
  }


}while(strlen(query_str)!=0);

            if(ANDflag==ORflag)
            {
                        ANDflag=0;
                        ORflag=1;
            }

  if(debug)
            printf("Keywords *%s* Input *%s* Output *%s* AND %d OR
%d\n",keywords,inputParam,outputParam,ANDflag,ORflag);

  if(strlen(keywords)==0 && strlen(inputParam)==0 && strlen(outputParam)==0)
            {
                        printf("All the fields can't be null\n");
                        printf("</html>");
                        exit(1);
            }


            memset(resultsend,'\0',256);
     memset(db_String,'\0',256);

    /*  send Multicast message to get the db_String */
     memset ((void *)&msend, 0, sizeof(Message_t));

     memset(msend.msg,'\0',MaxMsg);
     msend.type=htonl(DBASE_QUERY);
     msend.App=htonl(WSR);
     retcode = sendMulticast(&msend,resultsend);
```

```
        if ( retcode ==0)
        {
              strcpy(db_String,resultsend);
        }
        else
        {
              printf("Error in searching services. Contact Admininstrator\n");
              printf("</html>");
              exit(1);
        }

              sprintf(query,"SELECT DISTINCT A.%s, A.%s FROM  %s A, %s B WHERE
",WSR_SERVICE_NAME,WSR_SERVICE_SERVICEID,WSR_SERVICE,WSR_FUNCTION
S);
              if(strlen(keywords))
        {

              temp = strtok(keywords,",");

               while(temp!=NULL)
              {
                    strcat(KeywordsSearchString,"$");
                    strcat(KeywordsSearchString,temp);
                    strcat(KeywordsSearchString,"%");

                    if((temp=strtok(NULL,","))!=NULL)
                    strcat(KeywordsSearchString," | ");

              }

              sprintf(SQLstat,"(CONTAINS(A.%s,'%s') >
0",WSR_SERVICE_KEYWORDS,KeywordsSearchString);
              strcat(query,SQLstat);
              sprintf(SQLstat," OR CONTAINS(A.%s,'%s') >
0",WSR_SERVICE_DESCRP,KeywordsSearchString);
              strcat(query,SQLstat);
              sprintf(SQLstat," OR CONTAINS(A.%s,'%s') >
0)",WSR_SERVICE_NAME,KeywordsSearchString);
              strcat(query,SQLstat);
                          searchflag=1;
         }

              if(strlen(inputParam))
        {
              temp = strtok(inputParam,",");

               while(temp!=NULL)
              {
                    strcat(InputPSearchString,"$");
                    strcat(InputPSearchString,temp);
                    strcat(InputPSearchString,"%");
```

```
                if((temp=strtok(NULL,","))!=NULL)
                strcat(InputPSearchString," | ");

            }
            if(searchflag)
            {
                if(ORflag)strcat(query," OR ");
                else if(ANDflag)strcat(query," AND ");
            }
            sprintf(SQLstat,"CONTAINS(B.%s,'%s') >
0",WSR_FUNCTIONS_INPUTPARAM,InputPSearchString);
            strcat(query,SQLstat);
            searchflag=1;

        }

             if(strlen(outputParam))
        {

            temp = strtok(outputParam,",");

             while(temp!=NULL)
            {
                strcat(OutputPSearchString,"$");
                strcat(OutputPSearchString,temp);
                strcat(OutputPSearchString,"%");

                if((temp=strtok(NULL,","))!=NULL)
                strcat(OutputPSearchString," | ");

            }

            if(searchflag)
            {
                if(ORflag)strcat(query," OR ");
                else if(ANDflag)strcat(query," AND ");
            }

            sprintf(SQLstat,"CONTAINS(B.%s,'%s') >
0",WSR_FUNCTIONS_OUTPUTPARAM,OutputPSearchString);
            strcat(query,SQLstat);
            searchflag=1;
        }

        if(debug)
                     printf("Query *%s*\n",query);

             noColumns=2;
        for(i=0;i<noColumns;++i)
        {
            columns[i]=(char **)malloc(sizeof(char**));

        }

        st = fetch(query,columns,noColumns);


        if(st==0 || st==-1)
        {
            printf("No services found that matched the criteria\n");

                     for(i=0;i<noColumns;++i)
            {
                free(*(columns[i]));
            }

        }
             else
             {
                     printf("The following services were found:\n");
                     printTwoCol(*(columns[0]),*(columns[1]),"Service Names","Service Ids");

                     for(i=0;i<noColumns;++i)
            {
                free(*(columns[i]));
            }
             }
             time(&endexectime);
             endcputime=clock();
        if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);


 printf("</pre></body></html>");
}

void printTwoCol(char *col1,char* col2,char *title1,char *title2)
{
            char *temp,*temp1,*value,*value1;
            int count=0,i;

            printf("\n  %-40s %s\n\n",title1,title2);
                           while((temp = strrchr(col1,','))!=NULL)
                           {
                                printf("%d.",++count);
                                *temp='\0';
                                if((temp=strrchr(col1,','))!=NULL)
                                {
                                     value=temp+1;
                                }
                                else
```

```
                                        value=col1;
                                temp1 = strrchr(col2,',');
                                 *temp1='\0';
                                 if((temp1=strrchr(col2,','))!=NULL)
                                 {
                                        value1=temp1+1;
                                 }
                                 else
                                        value1=col2;
printf("<a href=javascript:OpenWindow(\"/cgi-
bin/getServiceById?serviceid=%s\",20,0,\"%d\")>%s</a>",value1,count,value);
                                                for(i=0;i<(40-strlen(value));++i)printf(" ");
                                                        printf("\t%s\n",value1);
                        }
}
```

9) UpdateAccount.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
int debug;

void code(char *);

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

   char      *query_str;          /* input buffer.             */
   entry     ent;
   char      username[MAX_USER];
   char      password[MAX_PASS];
   char      address[MAX_ADDRESS];
   char      email[MAX_EMAIL];
   char      phone[MAX_PHONE];
   int       updatePass=0,updateAddress=0,updateEmail=0,updatePhone=0;
   char      newpassword[MAX_PASS];
   char      busid[64];
   char      SQLstat[256];
```

```
  int retcode,status,updateflag=0;
      char resultsend[256];
              char         transactionid[64];

 Message_t msend;
char query[256];  /* to form the query*/
      int st,i;
      char **columns[1]  /* Columns to be populated by the query */
      ,*temp;
      int noColumns=1;  /* number of columns to be populated by the query */
              clock_t clock(),startcputime,endcputime;
              time_t startexectime,endexectime;

 startcputime=clock();
  time(&startexectime);

   memset(username,'\0',MAX_USER);
  memset(password,'\0',MAX_PASS);
  memset(newpassword,'\0',MAX_PASS);
  memset(address,'\0',MAX_ADDRESS);
  memset(email,'\0',MAX_EMAIL);
  memset(phone,'\0',MAX_PHONE);
  memset(busid,'\0',64);
               memset(transactionid,'\0',64);

      /* Form the query to retrieve the unique business id for a given username and password */
      for(i=0;i<noColumns;++i)
      {
             columns[i]=(char **)malloc(sizeof(char**));

      }
      /* Call the function to query the database */

  /**************************/
  /* not buffer the stdout: */
  /**************************/
  setbuf(stdout, NULL);

  /*putenv("ORACLE_SID=beagle");*/

  printf("Content-type: text/html\n\n");
   printf("<html><pre>");
    printf("<title>Update Account</title>\n<body background=\"/webservices/grey.bmp\">");

  /*printf("*%s*\n",getenv("ORACLE_SID"));*/

  /**********************/
  /* read the cgi input: */
  /**********************/
  query_str = read_POST();
   /*strcpy(query_str,getenv("QUERY_STRING"));*/
              if(debug)
                          printf("Query String *%s*\n",query_str);

  /****************/
  /* get the session id*/
  /****************/

 do
 {
  getword(ent.val, query_str, '&');
  plustospace(ent.val);
  unescape_url(ent.val);
  getword(ent.name, ent.val, '=');

  if(!strcmp(ent.name,"username"))
      {
             if(strlen(ent.val)<MAX_USER)
                  strcpy(username,ent.val);
             else  strncpy(username,ent.val,MAX_USER-1);
      }
  else if(!strcmp(ent.name,"oldpassword"))
      {
              if(strlen(ent.val)<MAX_PASS)
                  strcpy(password,ent.val);
             else  strncpy(password,ent.val,MAX_PASS-1);
              code(password);
              }
 else if(!strcmp(ent.name,"password"))
      {
              if(strlen(ent.val)<MAX_PASS)
                  strcpy(newpassword,ent.val);
             else  strncpy(newpassword,ent.val,MAX_PASS-1);
              code(newpassword);
              }
  else if(!strcmp(ent.name,"address"))
      {
             if(strlen(ent.val)<MAX_ADDRESS)
                  strcpy(address,ent.val);
             else  strncpy(address,ent.val,MAX_ADDRESS-1);
      }
  else if(!strcmp(ent.name,"email"))
      {
             if(strlen(ent.val)<MAX_EMAIL)
                  strcpy(email,ent.val);
             else  strncpy(email,ent.val,MAX_EMAIL-1);
      }
  else if(!strcmp(ent.name,"phone"))
      {
             if(strlen(ent.val)<MAX_PHONE)
                 strcpy(phone,ent.val);
             else  strncpy(phone,ent.val,MAX_PHONE-1);
         }


     if(!strcmp(ent.name,"UpdatePass"))
     {
             if(!strcmp(ent.val,"true")) updatePass=1;
     }
     else if(!strcmp(ent.name,"UpdateEmail"))
     {
         if(!strcmp(ent.val,"true")) updateEmail=1;
     }
     else if(!strcmp(ent.name,"UpdateAddress"))
     {
         if(!strcmp(ent.val,"true")) updateAddress=1;
     }
     else if(!strcmp(ent.name,"UpdatePhone"))
     {
         if(!strcmp(ent.val,"true")) updatePhone=1;
     }
   }while(strlen(query_str)!=0);

             if(debug)
                     printf("Username *%s* Pass *%s* NewPass *%s* \n Address *%s* Email
 *%s* Phone *%s*\n",username,password,newpassword,address,email,phone);

             if(debug)
                     printf("UpPass %d UpAdd %d UpEmail %d UpPhone
 %d\n",updatePass,updateAddress,updateEmail,updatePhone);

             if(strlen(username)==0 || strlen(password)==0)
         {
             printf("Username or Password can't be null\n");
             printf("</pre></html>");
             exit(1);
         }

         memset(resultsend,'\0',256);
         memset(db_String,'\0',256);

        /*  send Multicast message to get the db_String */
         memset ((void *)&msend, 0, sizeof(Message_t));

         memset(msend.msg,'\0',MaxMsg);
         msend.type=htonl(DBASE_QUERY);
         msend.App=htonl(WSR);
         retcode = sendMulticast(&msend,resultsend);
         if ( retcode ==0)
         {
             strcpy(db_String,resultsend);
         }
```

```
else
{
       printf("Error in Updating Account. Contact Admininstrator\n");
        printf("</html>");
                exit(1);
        }

        status = authenticateUser(username,password,busid);

        if(debug)
                printf("Authentication status %d\n",status);
        if(status!=0)
{
                printf("%s\n",busid);
                printf("</html>");
       exit(1);
}



memset ((void *)&msend, 0, sizeof(Message_t));
memset(msend.msg,'\0',MaxMsg);

msend.type=htonl(UPDATE);
msend.App=htonl(WSR);


        sprintf(transactionid,"%ld%s",startexectime,busid);

          sprintf(msend.msg,"%s,UPDATE %s SET ",transactionid,WSR_AUTH);

if(updatePass)
{
       sprintf(SQLstat,"%s = '%s'",WSR_AUTH_WORD,newpassword);
       strcat(msend.msg,SQLstat);
       updateflag=1;
}

if(updateAddress)
{
       if(updateflag)strcat(msend.msg,", ");
       sprintf(SQLstat,"%s = '%s'",WSR_AUTH_ADD,address);
       strcat(msend.msg,SQLstat);
       updateflag=1;
}

        if(updateEmail)
{
       if(updateflag)strcat(msend.msg,", ");
       sprintf(SQLstat,"%s = '%s'",WSR_AUTH_EMAIL,email);
       strcat(msend.msg,SQLstat);
```

```
        updateflag=1;
    }

    if(updatePhone)
    {
        if(updateflag)strcat(msend.msg,", ");
        sprintf(SQLstat,"%s = '%s'",WSR_AUTH_PHONE,phone);
        strcat(msend.msg,SQLstat);
        updateflag=1;
    }

    sprintf(SQLstat," WHERE %s='%s'",WSR_AUTH_BUSID,busid);
        strcat(msend.msg,SQLstat);


        if(debug)
        printf("Update Query *%s*\n",msend.msg);
    if(updateflag)
        {
                if(sendMulticast(&msend,resultsend))
        {
            printf("Error in updating data. Contact administrator");
        }

        else
        {
            if(!strcmp(resultsend,"OK"))
            {
                printf("Data updated for the given account");
            }
            else
            {
                printf("Error in updating data. Contact administrator");
            }
        }

    }
        time(&endexectime);
        endcputime=clock();
    if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

  printf("</pre></body></html>");
  fflush(stdout);
}
```

10) UpdateService.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "cgi.h"
#include "mcast.h"
#include "database.h"

char db_String[256];
char *read_POST();
                int debug;

void code(char *);

#ifdef DEBUGL
debug=1;
#else
debug=0;
#endif

main(){

   char      *query_str;            /* input buffer.              */
   char      *temp;
   entry     ent;
   char     username[MAX_USER];
   char     password[MAX_PASS];
   char     serviceName[MAX_SERVICENAME];
   char     serviceDesc[MAX_SERVICEDESC];
   char     keywords[MAX_KEYWORDS];
   char     docUrl[MAX_DOCURL];
   char     accessPointsAdd[MAX_ACCESSPOINTS];
   char     accessPointsDelete[MAX_ACCESSPOINTS];
   char      serviceid[64];
   char     functionsAdd[256];
   char           functionsDelete[256];
   char     outputParam[512];
   char     inputParam[512];
   int
updateName=0,updateDesc=0,updateKeywords=0,updateDocUrl=0,addAccessPoints=0,deleteAc
cessPoints=0,addFunctions=0,
                              deleteFunctions=0;
   char      busid[64];
   char      SQLstat[256];
   char     name[16][64];
   char     input[16][128];
   char     output[16][128];
   int retcode,status,updateflag=0;
        char resultsend[256];
  Message_t msend;
```

```
     int st,i,noColumns,count=0;
  char       **columns[1];
  char        transactionid[64];
  clock_t clock(),startcputime,endcputime;
  time_t startexectime,endexectime;

 startcputime=clock();
  time(&startexectime);

   memset(username,'\0',MAX_USER);
  memset(password,'\0',MAX_PASS);
    memset(serviceName,'\0',MAX_SERVICENAME);
  memset(serviceDesc,'\0',MAX_SERVICEDESC);
  memset(keywords,'\0',MAX_KEYWORDS);
  memset(docUrl,'\0',MAX_DOCURL);
  memset(accessPointsAdd,'\0',MAX_ACCESSPOINTS);
  memset(accessPointsDelete,'\0',MAX_ACCESSPOINTS);
  memset(serviceid,'\0',64);
   memset(functionsAdd,'\0',256);
   memset(functionsDelete,'\0',256);
  memset(inputParam,'\0',512);
  memset(outputParam,'\0',512);
  memset(transactionid,'\0',64);
  /***************************/
  /* not buffer the stdout: */
  /***************************/
  setbuf(stdout, NULL);

  printf("Content-type: text/html\n\n");
   printf("<html><pre>");
              printf("<title>Update Service</title>\n<body
background=\"/webservices/grey.bmp\">");


  /************************/
  /* read the cgi input: */
  /************************/
  query_str = read_POST();



   printf("<pre>");
    if(debug) printf("Query String *%s*\n",query_str);

 do
 {
  getword(ent.val, query_str, '&');
  plustospace(ent.val);
  unescape_url(ent.val);
  getword(ent.name, ent.val, '=');
 if(!strcmp(ent.name,"username"))
    {
          if(strlen(ent.val)<MAX_USER)
                 strcpy(username,ent.val);
          else  strncpy(username,ent.val,MAX_USER-1);
    }
else if(!strcmp(ent.name,"password"))
    {
           if(strlen(ent.val)<MAX_PASS)
                 strcpy(password,ent.val);
          else  strncpy(password,ent.val,MAX_PASS-1);
                        code(password);
    }
else if(!strcmp(ent.name,"serviceName"))
    {
          if(strlen(ent.val)<MAX_SERVICENAME)
                 strcpy(serviceName,ent.val);
          else  strncpy(serviceName,ent.val,MAX_SERVICENAME-1);
    }
else if(!strcmp(ent.name,"serviceDesc"))
    {
           if(strlen(ent.val)<MAX_SERVICEDESC)
                 strcpy(serviceDesc,ent.val);
          else  strncpy(serviceDesc,ent.val,MAX_SERVICEDESC-1);
    }
else if(!strcmp(ent.name,"keywords"))
    {
          if(strlen(ent.val)<MAX_KEYWORDS)
                 strcpy(keywords,ent.val);
          else  strncpy(keywords,ent.val,MAX_KEYWORDS-1);
    }
else if(!strcmp(ent.name,"docUrl"))
    {
          if(strlen(ent.val)<MAX_DOCURL)
                 strcpy(docUrl,ent.val);
          else  strncpy(docUrl,ent.val,MAX_DOCURL-1);
    }
else if(!strcmp(ent.name,"accessPointsAdd"))
    {
                        if(strlen(ent.val)<MAX_ACCESSPOINTS)
                                    strcpy(accessPointsAdd,ent.val);
                        else  strncpy(accessPointsAdd,ent.val,MAX_ACCESSPOINTS-1);
           }
else if(!strcmp(ent.name,"accessPointsDelete"))
    {
                        if(strlen(ent.val)<MAX_ACCESSPOINTS)
                                    strcpy(accessPointsDelete,ent.val);
                        else  strncpy(accessPointsDelete,ent.val,MAX_ACCESSPOINTS-1);
           }
else if(!strcmp(ent.name,"serviceid"))
    {
```

```
                        if(strlen(ent.val)<64)
                                strcpy(serviceid,ent.val);
                        else  strncpy(serviceid,ent.val,64-1);
              }
else if(!strcmp(ent.name,"functionsDelete"))
    {
                        if(strlen(ent.val)<256)
                                strcpy(functionsDelete,ent.val);
                        else  strncpy(functionsDelete,ent.val,256-1);
              }
else if(!strcmp(ent.name,"func_name_all"))
    {
          if(strlen(ent.val)<256)
                strcpy(functionsAdd,ent.val);
          else  strncpy(functionsAdd,ent.val,256-1);
    }
else if(!strcmp(ent.name,"func_input_all"))
    {
          if(strlen(ent.val)<512)
                strcpy(inputParam,ent.val);
          else  strncpy(inputParam,ent.val,512-1);
    }
else if(!strcmp(ent.name,"func_output_all"))
    {
          if(strlen(ent.val)<512)
               strcpy(outputParam,ent.val);
          else  strncpy(outputParam,ent.val,512-1);
    }



 if(!strcmp(ent.name,"UpdateName"))
{
          if(!strcmp(ent.val,"true")) updateName=1;
}
else if(!strcmp(ent.name,"UpdateDesc"))
{
    if(!strcmp(ent.val,"true")) updateDesc=1;
}
else if(!strcmp(ent.name,"UpdateKeywords"))
{
    if(!strcmp(ent.val,"true")) updateKeywords=1;
}
else if(!strcmp(ent.name,"UpdateDocUrl"))
{
    if(!strcmp(ent.val,"true")) updateDocUrl=1;
}
else if(!strcmp(ent.name,"AddAccessPoints"))
{
    if(!strcmp(ent.val,"true")) addAccessPoints=1;
}
```

```
  else if(!strcmp(ent.name,"DeleteAccessPoints"))
  {
     if(!strcmp(ent.val,"true")) deleteAccessPoints=1;
  }
   else if(!strcmp(ent.name,"DeleteFunctions"))
  {
     if(!strcmp(ent.val,"true")) deleteFunctions=1;
  }
   else if(!strcmp(ent.name,"AddFunctions"))
  {
     if(!strcmp(ent.val,"true")) addFunctions=1;
  }


 }while(strlen(query_str)!=0);

            if(debug)
            {
                        printf("Username *%s* Pass *%s* Service Name *%s*\n Desc *%s* Id
*%s*\n",username,password,serviceName,serviceDesc,serviceid);

                        printf("Keywords *%s* DocUrl *%s*\n",keywords,docUrl);

                        printf("AccessPointsAdd *%s* Delete
*%s*\n",accessPointsAdd,accessPointsDelete);
                        printf("FunctionsAdd *%s*
FunctionsDelete*%s*\n",functionsAdd,functionsDelete);
                        printf("Input *%s* Output *%s*\n",inputParam,outputParam);
                        printf("UpName %d UpDesc %d UpKeywords %d UpDocUrl %d
\n",updateName,updateDesc,updateKeywords,updateDocUrl);
                        printf("AddAccess %d DeleteAccess
%d\n",addAccessPoints,deleteAccessPoints);
                        printf("AddFunctions %d DeleteFunctions
%d\n",addFunctions,deleteFunctions);
            }
            if(strlen(username)==0 || strlen(password)==0 || strlen(serviceid)==0)
      {
            printf("Username or Password or Serviceid can't be null\n");
            printf("</pre></html>");
            exit(1);
      }

            if(updateName==0 && updateDesc==0 && updateKeywords==0 &&
updateDocUrl==0 && addAccessPoints==0 && deleteAccessPoints==0 && addFunctions ==0
&& deleteFunctions==0)
            {
                        printf("No field set to be updated\n");
            printf("</pre></html>");
            exit(1);
            }
```

```
        memset(resultsend,'\0',256);
        memset(db_String,'\0',256);

       /*  send Multicast message to get the db_String */
        memset ((void *)&msend, 0, sizeof(Message_t));

        memset(msend.msg,'\0',MaxMsg);
        msend.type=htonl(DBASE_QUERY);
        msend.App=htonl(WSR);
        retcode = sendMulticast(&msend,resultsend);
        if ( retcode ==0)
        {
              strcpy(db_String,resultsend);
        }
        else
        {
              printf("Error in Updating Service. Contact Admininstrator\n");
               printf("</html>");
                           exit(1);
               }

               status = authenticateUser(username,password,busid);
               if(debug)
                           printf("Authentication status %d\n",status);
               if(status!=0)
        {
                           printf("%s\n",busid);
                           printf("</html>");
              exit(1);
        }

               sprintf(SQLstat,"SELECT serviceid FROM %s WHERE %s = '%s' AND %s
='%s'",WSR_BUSINESS,WSR_BUSINESS_BUSID,busid,WSR_BUSINESS_SERVICEID,serv
iceid);
        noColumns=1;
        for(i=0;i<noColumns;++i)
        {
              columns[i]=(char **)malloc(sizeof(char**));

        }

        st = fetch(SQLstat,columns,noColumns);

        if(st==0 || st==-1)
        {
              if(st==0)
               printf("Service %s not found for the given user %s",serviceid,username);
              else
                                      printf("Error in updating service. Contact Administrator\n");

              for(i=0;i<noColumns;++i)
```

```
          {
                free(*(columns[i]));
          }
     exit(1);
     }

     for(i=0;i<noColumns;++i)
          {
                free(*(columns[i]));
          }

     sprintf(transactionid,"%d%s",startexectime,busid);

memset ((void *)&msend, 0, sizeof(Message_t));
memset(msend.msg,'\0',MaxMsg);

msend.type=htonl(UPDATE);
msend.App=htonl(WSR);
sprintf(msend.msg,"%s, BEGIN ",transactionid);

if(updateName)
{
                sprintf(SQLstat,"UPDATE %s SET ",WSR_SERVICE);
                strcat(msend.msg,SQLstat);
          sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_NAME, serviceName);
                 strcat(msend.msg,SQLstat);
     updateflag=1;
}

if(updateDesc)
{
     if(updateflag)strcat(msend.msg,", ");
                else {
                           sprintf(SQLstat,"UPDATE %s SET ",WSR_SERVICE);
                           strcat(msend.msg,SQLstat);
                     }
     sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_DESCRP, serviceDesc);
                strcat(msend.msg,SQLstat);
     updateflag=1;
}

     if(updateKeywords)
{
     if(updateflag)strcat(msend.msg,", ");
                else {
          sprintf(SQLstat,"UPDATE %s SET ",WSR_SERVICE);
          strcat(msend.msg,SQLstat);
        }
                sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_KEYWORDS, keywords);
                strcat(msend.msg,SQLstat);
     updateflag=1;
```

```
        }

        if(updateDocUrl)
        {
            if(updateflag)strcat(msend.msg,", ");
                     else {
                sprintf(SQLstat,"UPDATE %s SET ",WSR_SERVICE);
                strcat(msend.msg,SQLstat);
              }
                     sprintf(SQLstat,"%s = '%s'",WSR_SERVICE_DOCURL, docUrl);
                     strcat(msend.msg,SQLstat);
            updateflag=1;
        }

            if(updateflag)
            {
                     sprintf(SQLstat," WHERE %s = 
'%s';",WSR_SERVICE_SERVICEID,serviceid);
            strcat(msend.msg,SQLstat);
            }

            if(addAccessPoints)
        {
            if((temp=strtok(accessPointsAdd,","))!=NULL)
            {
                sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s'); 
",WSR_ACCESSPOINTS,serviceid,temp);
                strcat(msend.msg,SQLstat);
            }
            while((temp=strtok(NULL,","))!=NULL)
            {
                sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s'); 
",WSR_ACCESSPOINTS,serviceid,temp);
                strcat(msend.msg,SQLstat);
            }

            updateflag=1;
        }

            if(deleteAccessPoints)
        {
            if((temp=strtok(accessPointsDelete,","))!=NULL)
            {
                sprintf(SQLstat,"DELETE FROM %s WHERE %s = 
'%s';",WSR_ACCESSPOINTS,WSR_ACCESSPOINTS_URL,temp);
                              strcat(msend.msg,SQLstat);
            }
            while((temp=strtok(NULL,","))!=NULL)
            {
                sprintf(SQLstat,"DELETE FROM %s WHERE %s = 
'%s';",WSR_ACCESSPOINTS,WSR_ACCESSPOINTS_URL,temp);
```

```
                                strcat(msend.msg,SQLstat);
            }

            updateflag=1;
        }

            if(deleteFunctions)
            {
                    if((temp=strtok(functionsDelete,","))!=NULL)
            {
                sprintf(SQLstat,"DELETE FROM %s WHERE %s =
'%s';",WSR_FUNCTIONS,WSR_FUNCTIONS_NAME,temp);
                                strcat(msend.msg,SQLstat);
            }
            while((temp=strtok(NULL,","))!=NULL)
            {
                sprintf(SQLstat,"DELETE FROM %s WHERE %s =
'%s';",WSR_FUNCTIONS,WSR_FUNCTIONS_NAME,temp);
                                strcat(msend.msg,SQLstat);
            }

            updateflag=1;
            }

            if(addFunctions)
            {
                    count=0;
            temp=strtok(functionsAdd,"|");
            while(temp!=NULL)
            {
            strcpy(name[count++],temp);
            temp=strtok(NULL,"|");
            }

                count=0;
            temp=strtok(inputParam,"|");
            while(temp!=NULL)
            {
            if(strcmp(temp,"NULL"))
                    strcpy(input[count++],temp);
            else strcpy(input[count++],"");
            temp=strtok(NULL,"|");
            }

                count=0;
            temp=strtok(outputParam,"|");
            while(temp!=NULL)
            {
                if(strcmp(temp,"NULL"))
                strcpy(output[count++],temp);
                else strcpy(output[count++],"");
                temp=strtok(NULL,"|");
            }


                    for(i=0;i<count;++i)
            {

                        if(strcmp(name[i],"NULL"))
            {

                 sprintf(SQLstat,"INSERT INTO %s VALUES ('%s','%s','%s','%s');
",WSR_FUNCTIONS,serviceid,name[i],input[i],output[i]);
                strcat(msend.msg,SQLstat);
            }
            }
            updateflag=1;

            }
            strcat(msend.msg," END;");
      if(debug)
                    printf("Query *%s*\n",msend.msg);

      if(updateflag)
            {
                    if(sendMulticast(&msend,resultsend))
            {
                  printf("Error in updating data. Contact administrator");
            }

            else
            {
                  if(!strcmp(resultsend,"OK"))
                  {
                        printf("Data updated for the given service");
                  }
                  else
                  {
                        printf("Error in updating data. Contact administrator");
                  }
            }

      }
            time(&endexectime);
            endcputime=clock();
      if(debug)printf("Time taken to execute cpu %ld exec %ld\n",endcputime-
startcputime,endexectime-startexectime);

  printf("</body></html>");
  fflush(stdout);
}
```

Web Service Interface

A) WsrAdminSoapServer

1) WsrAdminSoap.java

```
import java.io.*;
import java.util.*;
import java.lang.*;
import electric.registry.Registry;
import electric.util.Base64;


public class wsrAdminSoap
{

            native String createAcc(createAccountData c);
            native String createSer(createServiceData s);
            native String updateAcc(updateAccountData u);
            native String updateSer(updateServiceData u);
            native String deleteSer(deleteServiceData d);
            public wsrAdminSoap()
            {

            }

             public String createAccount(createAccountData c)
      {
              String result;
                      System.out.println(c.Username);
              System.out.println(c.Word);
              System.out.println(c.Address);
              System.out.println(c.Email);
              System.out.println(c.PhoneNo);
                      if(c.Username==null || c.Word==null || c.Username.length()==0 ||
c.Word.length()==0)
                      return "Either Username  or Password cannot be null";


                      if(c.Address==null)
              c.Address="";
                      if(c.PhoneNo==null)
                      c.PhoneNo="";
                      if(c.Email==null)
              c.Email="";
                      result = createAcc(c);

                      return result;
      }

            public String updateAccount(updateAccountData u)
```

```
        {
                String result;
                        int doupdate=0;
                        System.out.println(u.Username);
                System.out.println(u.UpdateWord);
                System.out.println(u.UpdateAddress);
                System.out.println(u.UpdateEmail);
                System.out.println(u.UpdatePhoneNo);

                        if(u.Username==null || u.Word==null || u.Username.length()==0 ||
u.Word.length()==0)
                return "Username or Password cannot be null";

                        if(u.UpdateWord ==true)
                {
                      if(u.NewWord==null || u.NewWord.length()==0)
                        return "New Password cannot be null";
                                doupdate=1;
                        }else u.UpdateWord=false;

                        if(u.UpdateAddress ==true)
                {
                      if(u.Address==null)
                         u.Address="";
                                doupdate=1;
                }else u.UpdateAddress=false;

                        if(u.UpdatePhoneNo ==true)
                {
                      if(u.PhoneNo==null)
                       u.PhoneNo="";
                                doupdate=1;
                }else u.UpdatePhoneNo=false;

                        if(u.UpdateEmail ==true)
                {
                      if(u.Email==null)
                       u.Email="";
                                doupdate=1;
                }else u.UpdateEmail=false;

                        if(doupdate==0)
                        return "No data to be updated";

                        result = updateAcc(u);
                return result;
        }

              public String createService(createServiceData s)
        {
                        int i;
```

```
            String result;
        System.out.println(s.Username);
        System.out.println(s.Word);
        System.out.println(s.ServiceName);
        System.out.println(s.ServiceKeywords);
        System.out.println(s.ServiceCategory);

            if(s.Username==null || s.Word==null || s.ServiceName==null ||
s.ServiceName.length() ==0 || s.Username.length()==0 || s.Word.length()==0)
        return "Username or Password or Service Name cannot be null";

            if(s.ServiceKeywords==null)
            s.ServiceKeywords="";
            if(s.ServiceCategory==null)
        s.ServiceCategory="";
            if(s.ServiceDesc==null)
        s.ServiceDesc="";
            if(s.ServiceDocUrl==null)
        s.ServiceDocUrl="";

            if(s.ServiceAccessPoints==null)
            {
                s.ServiceAccessPoints = new String[0];
            }


            for(i=0;i<s.ServiceAccessPoints.length;++i)
            {
                if(s.ServiceAccessPoints[i]==null)
                {
                s.ServiceAccessPoints[i]="";
                System.out.println(s.ServiceAccessPoints[i]);
                }
            }

            if(s.ServiceFunctions==null)
            {
                s.ServiceFunctions = new functions[0];
            }

            for(i=0;i<s.ServiceFunctions.length;++i)
            {
                if(s.ServiceFunctions[i]!=null)
                {
                    if(s.ServiceFunctions[i].Name==null)
                    {
                    s.ServiceFunctions[i].Name="";
                    System.out.println(s.ServiceFunctions[i].Name);
                }
                    if(s.ServiceFunctions[i].InputParam==null)
            {
```

```
                                        s.ServiceFunctions[i].InputParam="";
                                        System.out.println(s.ServiceFunctions[i].InputParam);
                                }
                                        if(s.ServiceFunctions[i].OutputParam==null)
                        {
                                        s.ServiceFunctions[i].OutputParam="";
                                        System.out.println(s.ServiceFunctions[i].OutputParam);
                                        }
                                }
                                else
                                {
                                        s.ServiceFunctions[i]=new functions();
                                        s.ServiceFunctions[i].Name="";
                                        s.ServiceFunctions[i].InputParam="";
                                        s.ServiceFunctions[i].OutputParam="";
                                }

                        }
                        result = createSer(s);
                        return result;
        }

            public String updateService(updateServiceData s)
        {
                        String result;
                        int i,doupdate=0;
                        if(s.Username==null || s.Word==null || s.ServiceId==null || s.ServiceId.length()
==0 || s.Username.length()==0 || s.Word.length()==0)
                    return "Username or Password or Service Id cannot be null";


                        if(s.UpdateServiceName==true)
                        {
                                if(s.ServiceName==null || s.ServiceName.length()==0)
                                return "Service Name cannot be null";
                                doupdate=1;
                        }else s.UpdateServiceName=false;

                        if(s.UpdateServiceKeywords==true)
                        {
                                if(s.ServiceKeywords==null)
                        s.ServiceKeywords="";
                                doupdate=1;
                        }else s.UpdateServiceKeywords=false;

                        if(s.UpdateServiceDesc ==true)
                    {
                          if(s.ServiceDesc==null)
                          s.ServiceDesc="";
                                doupdate=1;
                    }else s.UpdateServiceDesc=false;
        if(s.UpdateServiceDocUrl==true)
{
      if(s.ServiceDocUrl==null)
      s.ServiceDocUrl="";
              doupdate=1;
}else s.UpdateServiceDocUrl=false;

        if(s.UpdateServiceCategory==true)
{
      if(s.ServiceCategory==null)
      s.ServiceCategory="";
              doupdate=1;
}else s.UpdateServiceCategory=false;

        if(s.AddServiceAccessPoints==true)
        {
              if(s.ServiceAccessPointsToAdd==null)
        {
        s.ServiceAccessPointsToAdd = new String[0];
        }


        for(i=0;i<s.ServiceAccessPointsToAdd.length;++i)
        {
        if(s.ServiceAccessPointsToAdd[i]==null)
        {
        s.ServiceAccessPointsToAdd[i]="";
        System.out.println(s.ServiceAccessPointsToAdd[i]);
        }
                      else doupdate=1;
        }
        }else s.AddServiceAccessPoints=false;

        if(s.DeleteServiceAccessPoints==true)
{
      if(s.ServiceAccessPointsToDelete==null)
      {
            s.ServiceAccessPointsToDelete = new String[0];
      }


      for(i=0;i<s.ServiceAccessPointsToDelete.length;++i)
      {
            if(s.ServiceAccessPointsToDelete[i]==null)
            {
            s.ServiceAccessPointsToDelete[i]="";
            System.out.println(s.ServiceAccessPointsToDelete[i]);
            }
                      else doupdate=1;
      }
```

```
}else s.DeleteServiceAccessPoints=false;

        if(s.AddServiceFunctions ==true)
        {
                if(s.ServiceFunctionsToAdd==null)
          {
          s.ServiceFunctionsToAdd = new functions[0];
        }

        for(i=0;i<s.ServiceFunctionsToAdd.length;++i)
        {
        if(s.ServiceFunctionsToAdd[i]!=null)
        {
                if(s.ServiceFunctionsToAdd[i].Name==null)
                {
                s.ServiceFunctionsToAdd[i].Name="";
                System.out.println(s.ServiceFunctionsToAdd[i].Name);
                }
                                else doupdate=1;
                if(s.ServiceFunctionsToAdd[i].InputParam==null)
                {
                s.ServiceFunctionsToAdd[i].InputParam="";
                System.out.println(s.ServiceFunctionsToAdd[i].InputParam);
                }
                if(s.ServiceFunctionsToAdd[i].OutputParam==null)
                {
                s.ServiceFunctionsToAdd[i].OutputParam="";
                        System.out.println(s.ServiceFunctionsToAdd[i].OutputParam);
                }
        }
        else
        {
                s.ServiceFunctionsToAdd[i]=new functions();
                s.ServiceFunctionsToAdd[i].Name="";
                s.ServiceFunctionsToAdd[i].InputParam="";
                s.ServiceFunctionsToAdd[i].OutputParam="";
        }

        }

        }else s.AddServiceFunctions=false;

        if(s.DeleteServiceFunctions ==true)
{
      if(s.ServiceFunctionsToDelete==null)
      {
           s.ServiceFunctionsToDelete = new functions[0];
      }

      for(i=0;i<s.ServiceFunctionsToDelete.length;++i)
      {
```

```
                    if(s.ServiceFunctionsToDelete[i]!=null)
                    {
                        if(s.ServiceFunctionsToDelete[i].Name==null)
                        {
                        s.ServiceFunctionsToDelete[i].Name="";
                        System.out.println(s.ServiceFunctionsToDelete[i].Name);
                        }
                                else doupdate=1;
                        if(s.ServiceFunctionsToDelete[i].InputParam==null)
                        {
                        s.ServiceFunctionsToDelete[i].InputParam="";
                        System.out.println(s.ServiceFunctionsToDelete[i].InputParam);
                        }
                        if(s.ServiceFunctionsToDelete[i].OutputParam==null)
                        {
                        s.ServiceFunctionsToDelete[i].OutputParam="";
                         System.out.println(s.ServiceFunctionsToDelete[i].OutputParam);
                        }
                    }
                    else
                    {
                        s.ServiceFunctionsToDelete[i]=new functions();
                        s.ServiceFunctionsToDelete[i].Name="";
                        s.ServiceFunctionsToDelete[i].InputParam="";
                        s.ServiceFunctionsToDelete[i].OutputParam="";
                    }

                }

            }else s.DeleteServiceFunctions=false;

                if(doupdate==0)
                    return "No data to be updated";
                result = updateSer(s);
                return result;
        }

        public String deleteService(deleteServiceData d)
    {
            String result;
            System.out.println(d.Username);
            System.out.println(d.Word);
            System.out.println(d.ServiceId);
            if(d.Username==null || d.Word==null || d.ServiceId == null || d.ServiceId.length()==0 ||
d.Username.length()==0 || d.Word.length()==0)
            return "Username or Password or ServiceId cannot be null";

            result = deleteSer(d);
            return result;
    }
```

```
        static
    {
            String library =
"/usr/users/khambati/webservices/testexamples/wsrAdminSoapServer/libwsrAdminSoap.so"; // in
current directory
            try
            {
                    System.load( library );
            }
            catch( Exception e )
            {
                    System.out.println( "\nERROR: Unable to load " + library + "\nException: "
+ e + "\n" );
            }

        } // end static

}
```

B) WsrInquireSoapServer

1) WsrInquireSoap.java

```
#include <stdio.h>
#include <jni.h>
#include "wsrInquireSoapServer.h"
#include "wsrInquireSoap.h"
#include "wsrinclude.h"
#include "databasewsr.h"
#include "mcast.h"
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <libgen.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>

int debug;

#ifdef DEBUGL
debug=1;
#else
debug=0;
```

```
#endif

typedef struct
{
            char **Name;
            char **InputParam;
            char **OutputParam;
}ServiceFunctions;

char db_String[256];
FILE *flog;
int addModify(char *WSRid,int *process, char* Url);
void getServiceAccessPoint(char*,char*,int *);
void Interrupt(int i, siginfo_t *siginfo, void *context )
{
/* Code to execute on receivin a SIGUSR1 signal*/

}

JNIEXPORT void JNICALL Java_wsrInquireSoapServer_setSigUSR1
  (JNIEnv *env, jobject o)
{

 struct sigaction sigUSR1;
 int status;

 sigUSR1.sa_handler   = 0;
 sigUSR1.sa_sigaction = Interrupt;
 sigfillset( &sigUSR1.sa_mask );
 sigUSR1.sa_flags = SA_SIGINFO;

 status = sigaction( SIGUSR1, &sigUSR1, 0 );
 if (status != 0) {
  printf("Failed to set signal handler for signal SIGUSR1\n");
 }
 else
  printf("Signal handler set for signal SIGUSR1\n");
}

JNIEXPORT void JNICALL Java_wsrInquireSoapServer_CloseLog
  (JNIEnv *env , jobject o)
{

        fclose(flog);
}

JNIEXPORT void JNICALL Java_wsrInquireSoapServer_OpenLog
  (JNIEnv *env , jobject o)
{

        if((flog=fopen(LOGFILE,"a+"))!=NULL)
        {
              printf("Logfile %s opened for writing\n",LOGFILE);
        }
        else printf("Logfile %s could not be opened\n",LOGFILE);
}

JNIEXPORT void JNICALL Java_wsrInquireSoapServer_writeURL
  (JNIEnv *env , jobject o, jstring WSRidj, jstring URLj)
{

        const char *URL;
        const char *WSRid;
        FILE *fp;
        int pid=0;
        char Url[256];
        URL = (*env)->GetStringUTFChars(env, URLj, 0);
        WSRid = (*env)->GetStringUTFChars(env, WSRidj, 0);


        getInfo((char *)WSRid,&pid,Url);
        strcpy(Url,(char *)URL);
        if(debug)printf("IN writeURL:writing processid %d URL %s \n",pid,Url);
            addModify((char *)WSRid,&pid,Url);
}

JNIEXPORT jint JNICALL Java_wsrInquireSoapServer_finddatabase
  (JNIEnv *env, jobject o)
{
             Message_t msend;
             int           retcode;
            char          result[256];

            memset(result,'\0',256);
            memset(db_String,'\0',256);

            /* send Multicast message to get the db_String */
        memset ((void *)&msend, 0, sizeof(Message_t));

        memset(msend.msg,'\0',MaxMsg);
        msend.type=ntohl(DBASE_QUERY);

        retcode = sendMulticast(&msend,result);
            if ( retcode ==0)
        {
              strcpy(db_String,result);
              printf("Found a database to connect with having DbString %s\n",db_String);
            return retcode;
            }
            else
        {
              printf("Can't find a database to connect with\n");
            return retcode;
            }



}
int getStringFromObj(JNIEnv *env,jclass cls, jobject Data, char *param, char *value)
{
            jfieldID fid;
            jstring s;
            const char *result;

            fid = (*env)->GetFieldID(env,cls,param,"Ljava/lang/String;");
        s = (*env)->GetObjectField(env,Data,fid);
            result = (*env)->GetStringUTFChars(env, s, 0);
            strcpy(value,result);

            return 1;
}

int getBoolFromObj(JNIEnv *env,jclass cls, jobject Data, char *param)
{
        jfieldID fid;
        jboolean b;

        fid = (*env)->GetFieldID(env,cls,param,"Z");
        b = (*env)->GetBooleanField(env,Data,fid);

            return b;
}

int getArrayFromObj(JNIEnv *env,jclass cls, jobject Data, char *param, char **value,int
*arraylen)
{
        jfieldID fid;
        jobjectArray arr;
            jsize len;
            jstring s;
            const char *result;
            int i;

            fid = (*env)->GetFieldID(env,cls,param,"[Ljava/lang/String;");
        arr = (*env)->GetObjectField(env,Data,fid);

        len = (*env)->GetArrayLength(env,arr);

            for(i=0;i<len;++i)
            {
                        s = (*env)->GetObjectArrayElement(env,arr,i);
                        result = (*env)->GetStringUTFChars(env, s, 0);
                        value[i]=(char *)malloc(sizeof(char*));
                value[i]=(char *)malloc(sizeof(char)*(strlen(result)+1));
                strcpy(value[i],result);
            }
            *arraylen=len;
            return 1;
}

int getObjArrayFromObj(JNIEnv *env,jclass cls, jobject Data, char *param,ServiceFunctions
*value,int *arraylen)
{
        jfieldID fid,fid1;
        jobjectArray arr;
        jsize len;
        jstring s;
            jclass cls1;
        jobject j;
             const char *result;
        int i;

        fid = (*env)->GetFieldID(env,cls,param,"[Lfunctions;");
        arr = (*env)->GetObjectField(env,Data,fid);

        len = (*env)->GetArrayLength(env,arr);

        for(i=0;i<len;++i)
        {
              j = (*env)->GetObjectArrayElement(env,arr,i);
                        cls1 = (*env)->GetObjectClass(env,j);
                        fid1 = (*env)->GetFieldID(env,cls1,"Name","Ljava/lang/String;");
            s = (*env)->GetObjectField(env,j,fid1);
                        result = (*env)->GetStringUTFChars(env, s, 0);
            value->Name[i]=(char *)malloc(sizeof(char*));
              value->Name[i]=(char *)malloc(sizeof(char)*(strlen(result)+1));
              strcpy(value->Name[i],result);
        }
            for(i=0;i<len;++i)
        {
              j = (*env)->GetObjectArrayElement(env,arr,i);
              cls1 = (*env)->GetObjectClass(env,j);
              fid1 = (*env)->GetFieldID(env,cls1,"InputParam","Ljava/lang/String;");
              s = (*env)->GetObjectField(env,j,fid1);
              result = (*env)->GetStringUTFChars(env, s, 0);
              value->InputParam[i]=(char *)malloc(sizeof(char*));
              value->InputParam[i]=(char *)malloc(sizeof(char)*(strlen(result)+1));
              strcpy(value->InputParam[i],result);
        }
            for(i=0;i<len;++i)
        {
              j = (*env)->GetObjectArrayElement(env,arr,i);
              cls1 = (*env)->GetObjectClass(env,j);
              fid1 = (*env)->GetFieldID(env,cls1,"OutputParam","Ljava/lang/String;");
```

```
            s = (*env)->GetObjectField(env,j,fid1);
            result = (*env)->GetStringUTFChars(env, s, 0);
            value->OutputParam[i]=(char *)malloc(sizeof(char*));
            value->OutputParam[i]=(char *)malloc(sizeof(char)*(strlen(result)+1));
            strcpy(value->OutputParam[i],result);
        }

            *arraylen=len;

        return 1;
}

JNIEXPORT jstring JNICALL Java_wsrInquireSoapServer_getWSMAccessPoint
  (JNIEnv *env, jobject o)
{
            char query[256];
            char result[1024];
            jstring s;
            int noAccessPoints;

            sprintf(query,"SELECT A.%s FROM %s A, %s B  WHERE B.%s = '%s' AND B.%s
=A.%s",WSM_REPLICA_SERVICEURL,WSM_REPLICA,WSM_SERVICE,WSM_SERVICE
_NAME,WSM_WEBSERVICE_NAME,WSM_SERVICE_SERVICEID,WSM_REPLICA_SER
VICEID);

            getServiceAccessPoint(query,result,&noAccessPoints);

            if(strncmp(result,"ERROR",5))
                        s = (*env)->NewStringUTF(env,strtok(result,","));
            else
                        s = (*env)->NewStringUTF(env,result);
        return s;
}

void getServiceAccessPoint(char *query,char *accesspoint,int *noAccessPoints)
{
        int noColumns,i,st;
        char **columns[1];


        noColumns=1;
        for(i=0;i<noColumns;++i)
        {
            columns[i]=(char **)malloc(sizeof(char**));

        }

        st = fetch(query,columns,noColumns);
        *noAccessPoints=st;
            if(st>0)
               strcpy(accesspoint,*(columns[0]));
        else
                strcpy(accesspoint,"ERROR:");

        for(i=0;i<noColumns;++i)
        {
           free(*(columns[i]));
        }

             return;
}

JNIEXPORT jobject JNICALL Java_wsrInquireSoap_ServiceInfo
  (JNIEnv *env, jobject o, jstring serviceidj)
{

             jclass cls;
             jfieldID fid;
             jstring s,s1;
             jobjectArray args;
             char **columns[5],*temp;
             int noColumns,st,i;
             char query[256];
             const char *serviceid;


             serviceid = (*env)->GetStringUTFChars(env, serviceidj, 0);

              /* Finding the JAVA class to be called */
        cls = (*env)->FindClass(env, "Service");
        if (cls == 0) {
        fprintf(stderr, "Can't find Service class\n");
        }


        /* Creating an object of the class */
        jobject obj =  (*env)->AllocObject(env,cls);


             noColumns=3;

             for(i=0;i<noColumns;++i)
        {
               columns[i]=(char **)malloc(sizeof(char**));

        }


             sprintf(query,"SELECT %s,%s,%s FROM %s  where %s
='%s'",WSR_SERVICE_NAME,WSR_SERVICE_DESCRP,WSR_SERVICE_DOCURL,WSR_
SERVICE,WSR_SERVICE_SERVICEID,serviceid);
```

```
            fid = (*env)->GetFieldID(env,cls,"Name","Ljava/lang/String;");

            if(debug)printf("Service Info Query *%s*\n",query);
            st = fetch(query,columns,noColumns);

            if(st==0 || st==-1)
            {
                    s = (*env)->NewStringUTF(env,"ERROR:Error in Finding the service");
                    (*env)->SetObjectField(env,obj,fid,s);

                    for(i=0;i<noColumns;++i)
            {
                    free(*(columns[i]));
            }

                    return obj;

            }

            s = (*env)->NewStringUTF(env,strtok(*(columns[0]),","));
              (*env)->SetObjectField(env,obj,fid,s);

            fid = (*env)->GetFieldID(env,cls,"Description","Ljava/lang/String;");
             s = (*env)->NewStringUTF(env,strtok(*(columns[1]),","));
              (*env)->SetObjectField(env,obj,fid,s);

             fid = (*env)->GetFieldID(env,cls,"DocumentUrl","Ljava/lang/String;");
         s = (*env)->NewStringUTF(env,strtok(*(columns[2]),","));
              (*env)->SetObjectField(env,obj,fid,s);

            for(i=0;i<noColumns;++i)
        {
           free(*(columns[i]));
        }

            noColumns=1;

            for(i=0;i<noColumns;++i)
        {
              columns[i]=(char **)malloc(sizeof(char**));

        }


            /*get the accesspoints for the service */

            sprintf(query,"SELECT %s FROM %s where
%s='%s'",WSR_ACCESSPOINTS_URL,WSR_ACCESSPOINTS,WSR_ACCESSPOINTS_SER
VICEID,serviceid);
```

```
          if(debug)printf("Service AccessPoints Query *%s*\n",query);
          st = fetch(query,columns,noColumns);

          fid = (*env)->GetFieldID(env,cls,"Accesspoints","[Ljava/lang/String;");

      if(st==0 || st==-1)
      {
           s = (*env)->NewStringUTF(env,"No Accesspoints available for this service right
now");

                   args = (*env)->NewObjectArray(env,1,
                 (*env)->FindClass(env, "java/lang/String"), NULL);

                   (*env)->SetObjectArrayElement(env, args, (jsize)0, s);
          (*env)->SetObjectField(env,obj,fid,args);

            for(i=0;i<noColumns;++i)
            {
                  free(*(columns[i]));
            }

            return obj;

      }

          /* Creating an array */
      args = (*env)->NewObjectArray(env,st,
                 (*env)->FindClass(env, "java/lang/String"), NULL);


          s = (*env)->NewStringUTF(env,strtok(*(columns[0]),","));
          (*env)->SetObjectArrayElement(env, args, (jsize)0, s);

          i=1;
          while((temp=strtok(NULL,","))!=NULL)
          {
                   s = (*env)->NewStringUTF(env,temp);
          (*env)->SetObjectArrayElement(env, args, (jsize)i, s);
                   ++i;
          }

      (*env)->SetObjectField(env,obj,fid,args);

          for(i=0;i<noColumns;++i)
      {
         free(*(columns[i]));
      }

          return obj;
}
```

```
JNIEXPORT jobjectArray JNICALL Java_wsrInquireSoap_searchServices
  (JNIEnv *env, jobject o, jobject findServiceCriteriaj)
{


	int
Keywordsflag,InputParametersflag,OutputParametersflag,ORflag,ANDflag,searchflag=0;
	char Keywords[256];
      char InputParameters[256];
      char OutputParameters[256];
	char KeywordsSearchString[512];
	char InputPSearchString[512];
	char OutputPSearchString[512];
	char SQLstat[256];
	char query[1024];
	char *temp;
	char **columns[1];
	int noColumns,i,st;
	jobjectArray args;
	jclass cls;
	jstring s;

	memset(KeywordsSearchString,'\0',512);
	memset(InputPSearchString,'\0',512);
	memset(OutputPSearchString,'\0',512);

	cls = (*env)->GetObjectClass(env,findServiceCriteriaj);


	sprintf(query,"SELECT DISTINCT A.%s FROM  %s A, %s B WHERE
",WSR_SERVICE_SERVICEID,WSR_SERVICE,WSR_FUNCTIONS);

	ORflag = getBoolFromObj(env,cls,findServiceCriteriaj,"useORbetweenfields");
	ANDflag = getBoolFromObj(env,cls,findServiceCriteriaj,"useANDbetweenfields");

	Keywordsflag = getBoolFromObj(env,cls,findServiceCriteriaj,"SearchKeywords");
	if(Keywordsflag)
	{
		getStringFromObj(env,cls,findServiceCriteriaj,"Keywords",Keywords);

		temp = strtok(Keywords,",");

	      while(temp!=NULL)
	{
		strcat(KeywordsSearchString,"$");
		strcat(KeywordsSearchString,temp);
		strcat(KeywordsSearchString,"%");

		if((temp=strtok(NULL,","))!=NULL)
		strcat(KeywordsSearchString," | ");
```

```
        }

                sprintf(SQLstat,"(CONTAINS(A.%s,'%s') >
0",WSR_SERVICE_KEYWORDS,KeywordsSearchString);
                strcat(query,SQLstat);
                sprintf(SQLstat," OR CONTAINS(A.%s,'%s') >
0)",WSR_SERVICE_DESCRP,KeywordsSearchString);
                strcat(query,SQLstat);
                 sprintf(SQLstat," OR CONTAINS(A.%s,'%s') >
0)",WSR_SERVICE_NAME,KeywordsSearchString);
            strcat(query,SQLstat);
                searchflag=1;
         }

        InputParametersflag =
getBoolFromObj(env,cls,findServiceCriteriaj,"SearchInputParameters");
     if(InputParametersflag)
     {

        getStringFromObj(env,cls,findServiceCriteriaj,"InputParameters",InputParameters);

                temp = strtok(InputParameters,",");

           while(temp!=NULL)
           {
               strcat(InputPSearchString,"$");
               strcat(InputPSearchString,temp);
               strcat(InputPSearchString,"%");

               if((temp=strtok(NULL,","))!=NULL)
               strcat(InputPSearchString," | ");

           }

                if(searchflag)
                {
                        if(ORflag)strcat(query," OR ");
                        else if(ANDflag)strcat(query," AND ");
                }
                sprintf(SQLstat,"CONTAINS(B.%s,'%s') >
0",WSR_FUNCTIONS_INPUTPARAM,InputPSearchString);
            strcat(query,SQLstat);
            searchflag=1;

     }

        OutputParametersflag =
getBoolFromObj(env,cls,findServiceCriteriaj,"SearchOutputParameters");
     if(OutputParametersflag)
     {
        getStringFromObj(env,cls,findServiceCriteriaj,"OutputParameters",OutputParameters);

                printf("OUT *%s*",OutputParameters);
                temp = strtok(OutputParameters,",");

          while(temp!=NULL)
          {
               strcat(OutputPSearchString,"$");
               strcat(OutputPSearchString,temp);
               strcat(OutputPSearchString,"%");

               if((temp=strtok(NULL,","))!=NULL)
               strcat(OutputPSearchString," | ");

          }

          if(searchflag)
          {
               if(ORflag)strcat(query," OR ");
               else if(ANDflag)strcat(query," AND ");
          }

                sprintf(SQLstat,"CONTAINS(B.%s,'%s') >
0",WSR_FUNCTIONS_OUTPUTPARAM,OutputPSearchString);
          strcat(query,SQLstat);
          searchflag=1;
        }

        if(debug)printf("Search Services Query *%s*\n",query);

        noColumns=1;

     for(i=0;i<noColumns;++i)
     {
          columns[i]=(char **)malloc(sizeof(char**));

     }

     st = fetch(query,columns,noColumns);


        if(st==0 || st==-1)
     {
          s = (*env)->NewStringUTF(env,"No services found that matched the criteria");

          args = (*env)->NewObjectArray(env,1,
               (*env)->FindClass(env, "java/lang/String"), NULL);

          (*env)->SetObjectArrayElement(env, args, (jsize)0, s);
```

```
            for(i=0;i<noColumns;++i)
            {
                free(*(columns[i]));
            }

            return args;

        }


          /* Creating an array */
        args = (*env)->NewObjectArray(env,st,
                    (*env)->FindClass(env, "java/lang/String"), NULL);


        s = (*env)->NewStringUTF(env,strtok(*(columns[0]),","));
        (*env)->SetObjectArrayElement(env, args, (jsize)0, s);

        i=1;
        while((temp=strtok(NULL,","))!=NULL)
        {
            s = (*env)->NewStringUTF(env,temp);
            (*env)->SetObjectArrayElement(env, args, (jsize)i, s);
            ++i;
        }

        for(i=0;i<noColumns;++i)
        {
          free(*(columns[i]));
        }


          return args;


}

JNIEXPORT jobjectArray JNICALL Java_wsrInquireSoap_serviceAccessPoints
  (JNIEnv *env, jobject o, jstring serviceidj)
{

          const char *serviceid;
          char *temp;
          char query[256];
        char result[1024];
        jstring s;
          jobjectArray args;
          int i,noAccessPoints;

        serviceid = (*env)->GetStringUTFChars(env, serviceidj, 0);
        sprintf(query,"SELECT %s FROM %s  WHERE %s
='%s'",WSR_ACCESSPOINTS_URL,WSR_ACCESSPOINTS,WSR_ACCESSPOINTS_SERVI
CEID,serviceid);

    getServiceAccessPoint(query,result,&noAccessPoints);

        if(!strncmp(result,"ERROR",5))
    {
        s = (*env)->NewStringUTF(env,"No Accesspoints available for this service right
now");

        args = (*env)->NewObjectArray(env,1,
            (*env)->FindClass(env, "java/lang/String"), NULL);

        (*env)->SetObjectArrayElement(env, args, (jsize)0, s);

        return args;

    }


        /* Creating an array */
    args = (*env)->NewObjectArray(env,noAccessPoints,
            (*env)->FindClass(env, "java/lang/String"), NULL);


    s = (*env)->NewStringUTF(env,strtok(result,","));
    (*env)->SetObjectArrayElement(env, args, (jsize)0, s);

    i=1;
    while((temp=strtok(NULL,","))!=NULL)
    {
        s = (*env)->NewStringUTF(env,temp);
        (*env)->SetObjectArrayElement(env, args, (jsize)i, s);
        ++i;
    }

    return args;
}
```