INVOCATION OF WEB SERVICES IN GLYCOMICS PORTAL

by

SINGARAM SUNDAR

(Under the Direction of William York and John Miller)

ABSTRACT

The GlycomicsPortal is a web based portal intending to serve as a hub for the Glycomics community. It contains relevant Web services, workflow systems, software modules and database systems. The users can register, submit or view existing content. This research enhances the functionality of the web portal by adding the Web service invocation and automatic Web service population features to it. The primary focus of these features is to enable the users to directly invoke and execute all the SOAP and REST Web services registered within the portal without writing a separate client application. This is achieved by introducing a generic SOAP client that handles doc-literal SOAP and REST clients that handles REST Web services. These SOAP and REST clients are integrated within the portal to exploit its functionality.

INDEX WORDS: GlycomicsPortal, Web services, SOAP, REST, WSDL

INVOCATION OF WEB SERVICES IN GLYCOMICS PORTAL

by

SINGARAM SUNDAR

B.Tech, SRM University, India, 2009

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2012

INVOCATION OF WEB SERVICES IN GLYCOMICS PORTAL

by

SINGARAM SUNDAR

Major Professor: William York, John A. Miller

Committee: Krzysztof J. Kochut
Ismailcem Budak Arpinar

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
July 2012

DEDICATION

To God, my family and friends.

ACKNOWLEDGEMENTS

I am greatly thankful to my major professors, Dr. York and Dr. Miller for their continuous support and guidance during my study at UGA. I would like to thank Dr. York for his valuable advice, support and encouragement through all my work and time at UGA. I would like to thank Dr. Miller for his valuable suggestions, support, advice and encouragement all through my work. I would like to thank Rene Ranzinger for his continuous guidance, support and teaching all through my work. I am thankful to Dr. Kochut, who has introduced me to Web services, helped in understanding the concepts and built the foundation. I would also like to thank Dr. Arpinar for being a part of my work and time at UGA.

Finally, I would like to thank my colleagues who have helped me on this project: Alok Dhamanaskar and Micheal A. Cotterell.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Overview

Sophisticated computational tools are required to process and annotate the large, complex data sets that have become a hallmark of modern biology. Many of these tools are available as Web services, which perform diverse operations on distinct experimental data types. However, identifying the appropriate Web service to perform a specific task in a defined environment is still problematic. Although the basic functionality of a Web service can often be ascertained from descriptions such as those associated with each service in a repository, invoking a Web service in a way that provides enough information to evaluate its utility often requires the user to write customized client software. Thus, generic client software that allows the user to invoke the full functionality of diverse Web services could save a considerable amount of effort by individuals who merely want to evaluate or use the service.

The available Web services are registered in different repositories, which provide the information required to access them. In a typical scenario where there is no generic client software available as a part of the repository, users will have to gather the details provided about the Web service in the repository and write customized client software to consume the specific Web service. The alternative way to consume the service will be to

use the provided details in another generic client software. This thesis addresses this problem by bringing the repository that contains the registered Web services and the generic client software together. The Web services can be discovered using the repository and invoked using the generic client software from the same place. The work focuses on merging the Web service repository and the Web service invocation system to enable automatic Web service invocation from the repository, substantially reducing the effort required to invoke a Web service. It expands the functionality of the repository by automatically populating the Web services based on its definition and annotating Web service components with descriptions from the service provider that facilitate efficient use.

1.2 Motivation

Writing client software to invoke a Web service is a resource and time consuming process. There are several Web services available, however, in order for the users to be able to find the service based on their needs, they need to be able to test and evaluate it. Currently, the users are unable to do so without having to write a custom client or using another generic client to invoke and execute the service.

This thesis focuses on merging the Web service repository and the Web service invocation system together; to enable automatic Web service invocation from the repository. This result in substantially reducing the effort required for invoking a Web service.

CHAPTER 2

RELATED WORK

This section discusses several solutions related to discovering, registering, invoking, annotating and monitoring the web services. This chapter elaborates on the functionality and limitations of these systems.

2.1 Bio-Catalogue

The Bio catalogue [1] is a centralized registry of curated Life Science Web services. It allows the user to easily discover, register, annotate, monitor and use Web services. The Bio-catalogue system does not provide automatic Web service invocation functionality for the services registered in it and it also does not provide automatic population of Web services based on WSDL. This is a drawback compared to our system, which provides both functionalities at the same place.

2.2 EBI (European Bio Informatics Institute) System

European Bio Informatics Institute [2] is another system offering a Web service repository that contains Web services dealing with biological data. It allows programmatic access to various data resources and analytical tools. Even though this system contains various REST and SOAP services, it differs from our system in that it does not have a generic SOAP and REST client that can invoke any registered service.

2.3 Soap UI

SOAPUI [3] is a desktop based functional testing system that supports invocation of various formats of SOAP services. The WSDL for the SOAP service is provided manually and the system gives an interface in XML format for invoking any operation definied in the WSDL. In contrast to SOAPUI, our system provides the interface for service invocation directly from the repository through an intuitive, user-friendly approach that allows users without technical background to invoke and use the service.

2.4 Generic Web Based or desktop Clients

There are some generic web-based or standalone desktop clients like soapclient.com[4] that can invoke Web services when the definition file is provided. These generic clients support either SOAP or REST individually but our system supports both SOAP and REST.

The generic clients that are available can rarely be used as alternatives to the generic SOAP and REST client built into our system due to differences in software dependencies that are characteristic of each Web service. In addition, using a generic client forces our system to depend on the third party client, affecting reliability of our system. If we were to use a third party generic client software and if the client were to change for any reason by its host, our system would get directly affected. Also, these systems do not offer an API and hence HTML feeding is not a reliable option. Based on the mentioned limitations and drawbacks, using our own system instead of the available generic clients would be the most optimal choice. Using our own SOAP and REST

client, will enable us to seamlessly integrate them with the Glycomics portal and to exploit the benefits of both systems together.

2.5 WSDL-Based Automatic Test Case Generation for Web Services Testing

Xiaoying Bai, Wenli Dong, Wei-Tek Tsai and Yinong Chen, discuss and propose a way of generating test cases for Web services [5]. Web service have become an essential part of today's world and the number of Web services being published every day is growing at a very fast pace. However, one must realize that in order to maintain the quality of the services that are being published, invoked and integrated, test cases need to be generated at runtime. This paper [5] proposes a method to generate these test cases automatically. These test cases are generated with the help of WSDL files that have the information about the Web service. The approach that was taken for generating the test cases was based on two perspectives; test data generation and test operation generation. Once these are generated, three types of dependencies were defined including input dependency, output dependency, and input/output dependency. Finally, Service Test Specification files were generated in XML based files.

2.6 Daios: Efficient Dynamic Web Service Invocation

The Daios system stands for the Dynamic and Asynchronous Invocation of Services framework [6]. It is a message based service framework. It is based on RESTful services and supports SOA implementation, allowing dynamic invocation of SOAP/WSDL. With the help of the Daios system that is proposed in the system, users

can create stubless and dynamic service clients that are less strongly coupled to a specific service provider. This paper proposes a system (Daios) as a Web Service invocation front-end. It is a front end for SOAP/WSDL based services as well as the RESTful services. One of the advantages of this system is its ability to completely support dynamic invocations without any static components. A preliminary evaluation of the proposed system (Daios) was also done against several other systems including Apache WSIF, Apache Axis 2, Codehaus XFire, and Apache CXF. Figure 1 illustrates the architecture and framework of the proposed DAIOS system:

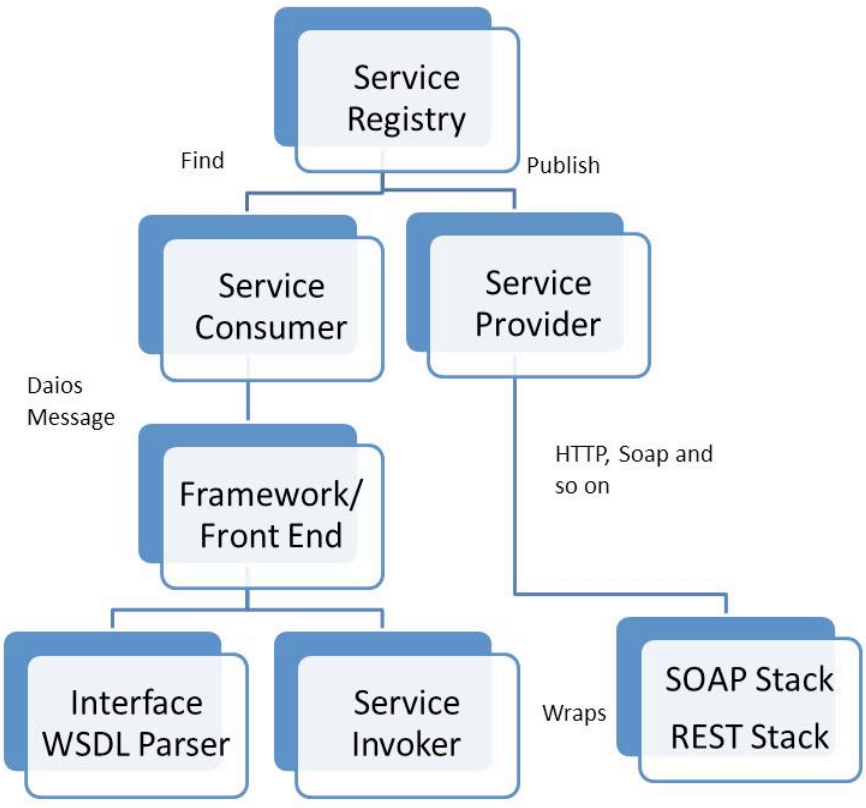

Figure 1: Daios overall architecture adapted from [6]

2.7 Web services Invocation Framework (WSIF)

In this paper, Matthew J. Duftler, Nirmal K. Mukhi, Aleksander Slominski and Sanjiva Weerawarana discusses and proposes a framework known as the Web service invocation framework (WSIF) allowing the application programmers to program in a protocol independent manner against an abstract description of service [7]. WSIF is implemented via an API, which is used to invoke the Web services. The system works using the WSDL files that describe the service, regardless of the nature of the system. The system enables the user to deal with abstract service representations rather than working closely with the SOAP API's. Using this system new bindings can be dynamically formed. WSIF is not up to date and was last updated in 2003. WSIF provisions two of the following different approaches:

1. This first approach deals with compiling the WSDL document to Java interface, implementation of the interface with the necessary Java types. The service can easily be accessed via the Java stubs that were generated by the implementation of the interface.

2. The second approach that was taken in this paper was to directly operate on WSDL documents. The key difference is that no compilation cycle is required when using this second approach.

Although there are several software tools available for discovering, registering, monitoring and using Web services, they either lack invocation functionality, generic SOAP or REST client or provide a limited SOAP or REST client. Our system provides a one-stop shop for Web service discovery and invocation with generic SOAP and REST client integrated with the GlycomicsPortal. This research brings the Web service repository (GlycomicsPortal) and the Web service Invocation system (SOAP and REST Generic Client) together to create a Web service Discovery and Invocation tool with Automatic Web service population.

CHAPTER 3

GLYCOMICS PORTAL

3.1 Overview

The Glycomics community does not have a one-stop place for finding relevant Web services, software modules, workflows and databases. The GlycomicsPortal is a solution to that requirement developed at the Complex Carbohydrate Research Center by Ranzinger and coworkers (http://glycomics.ccrc.uga.edu/GlycomicsPortal/) [15]. This portal makes it easy for many biologists and scientists to search and find Web services, workflows, databases and software modules. It is a centralized hub for glycomics related information.

This web portal contains Web services, Workflow systems, Software modules and Database systems related to Glycomics. It is open to the public so that anyone in this field can contribute to the knowledge source, populating the portal and making it more meaningful. This portal also has a user authentication system that has restrictions based on the user access level.

This portal is based on Client-Server architecture and a Model View Controller (MVC) interface. It has a database backend at the server side and a user interface at the client side that interacts with the server to perform all operations.

3.2 Architecture

- **PostgreSQL database layer:** PostgreSQL database [16] which is an open source object-relational database system is used as the database back end for the web portal.

- **Hibernate persistent layer:** Persistent layer is an object relational model representing tables in the database as classes, columns as objects and keys as sets. Hibernate [17] is a persistent system that provides a framework for object relational mapping (ORM) for Java. The database tables are basically accessed through Java classes and objects.

  Java bean classes [18] exist for every table in the database. A hibernate configuration XML file defines the mapping between the tables, columns and keys with Java class and objects for each table in the database.

- **Business logic layer:** The business logic for the portal is completely built using Java. This layer is responsible for all the logical operations in portal. The logic layer is split into three sub layers to perform its responsibilities.

- **Hibernate entity manager:** This layer contains all the logical code and interacts with the persistent layer to access the database (read and write). Every interaction with the hibernate layer to access the database is done within a transaction.
  Hibernate query language (HQL) [19] is used for interacting with the database from the entity manager.

- **Data transfers objects:** This layer contains classes and objects for transferring the data from the entity manager to the action classes. It secures and isolates the entity manager data from the action classes. It contains the getters and setters for the objects required at the action layer.

- **Action classes:** This layer contains action classes that handle all the events happening between the portal and the user. It sends data to the entity manager for various operations and receives data from the data transfer layer to display the user. It also contains the Java reflection for objects whose values are required at the front end.

- **Web Works framework:** Web works is a Java based web application framework that is built on top of Xwork [20]. It contains the web portals data flow between the data model and the front end. The entry point of the web application is web.xml, which redirects to the Web work's configuration xwork.xml that defines all the actions, its input parameter, interceptors and output template files. Input parameters contains the elements that the action takes in and output contains the location of the free marker template file that is displayed as a result of the execute method in the action.

- **Free marker template engine:** Free marker [21] is a tool that generates output for web pages for given html and Java objects. It is defined as .ftl files that contains HTML, Javascript and free marker code that represents Java objects at the action layer associated with this template. It is a dynamic tool for generating web pages and called as a result of an action from xwork.xml

- **Front end:** The front end that interacts with the user is designed using HTML, Javascript, jQuery, CSS and free marker. The collection of free marker template files dynamically generate the front end web pages based on the action and events.

3.3 System

The GlycomicsPortal allows the user to submit either one of the following:

1. Web Service
2. Workflow System

3. Database System
4. Software Module

3.3.1 Web service

A Web service can be described as a program located at a remote server that does some operation, which is invoked by a client at a different location by passing the inputs required for executing the program [22]. This makes it easy to perform various operations, as we need not have the programs implemented locally. Instead, we make a call to that program through the appropriate protocol and get the result back. This also saves resources locally and is very useful for resource intensive programs.

Processing of glycomics data similarly involves diverse computational operations. This portal is a centralized repository for finding Web services related to glycomics which are submitted by various people who have implemented the services and make them publicly available.

3.3.2 Workflow System

A workflow consists of a sequence of steps or processes. Each step or process follows the previous step and must end before the next step/process can begin. A workflow management system defines a series of tasks to be performed for obtaining a particular output [23]. Many different workflows can be imagined for the analysis of glycomics data. Each of these would perform a set of operations with a given input to achieve a defined result. The workflow contains information about its framework, inputs, outputs and portal entries associated with it. Workflows developed by portal users for glycomics analysis can be submitted to the portal.

3.3.3 Database System

A database is an organized collection of data. Databases that contain a considerable amount of information about glycomics exist. The experimental data in glycomics databases are available to the biologists. The databases are for instance categorized as carbohydrates structure database or enzyme related database. The

databases contain information about the content, supported search for the data and semantics associated with it.

3.3.4 Software Module

A software module may perform one or more operations. It may be installed to the local system or used as a client to access information on a server. Glycomics has many software applications that perform various tasks related to it. The software modules contain information about the programming language, requirements, snapshots, download link and are submitted by developers who have created them.

The GlycomicsPortal home page is the base page of the system that is presented when the user first enters the portal. Figure 2 represents the Glycomics portal home page. The user can navigate through the portal to any type of entry from the menu bar. The recently added entries are displayed under 'Recent Items' and the total numbers of entries based on different types are displayed under 'Summary'.

Figure 2: GlycomicsPortal home page

The user clicks on the entry to open the entry page that displays entry specific information. Glycome DB is a database registered with GlycomicsPortal under the

category ‘Database’. Figure 3 represents the Glycome DB entry page that displays relevant information submitted by the user.

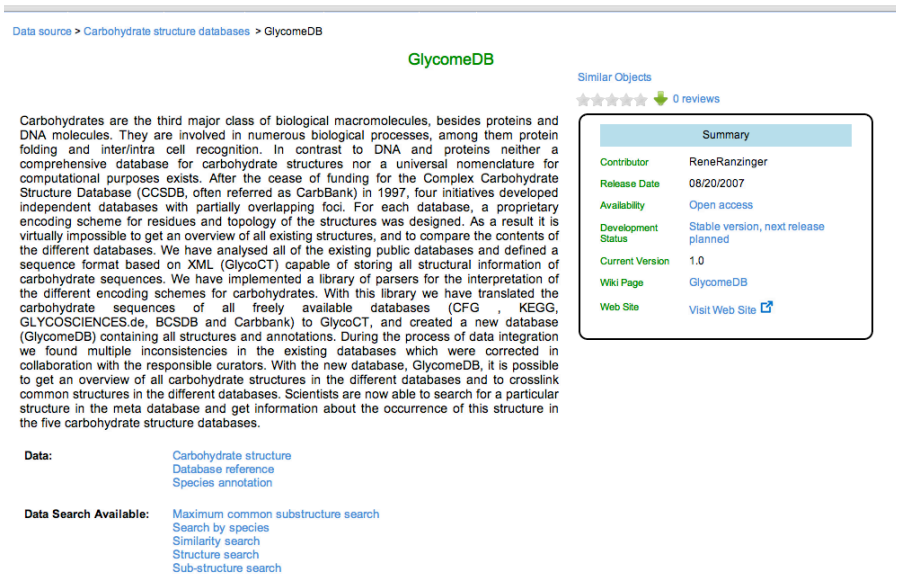

Figure 3: Glycome DB database entry page

3.4 Entry Submission

The GlycomicsPortal has a user authentication system that allows the user to register with the portal and provides a user name and password that the user uses to log into the system. Registered users are allowed to submit their Web services, databases, software module and workflow system related to Glycomics. The user follows a specific series of steps to submit a new entry in the portal. These involve specification of generic information about an entry, entry specific information, data related to the entry and more details if needed. Figure 4 illustrates the workflow of the system for a new entry submission by the user:

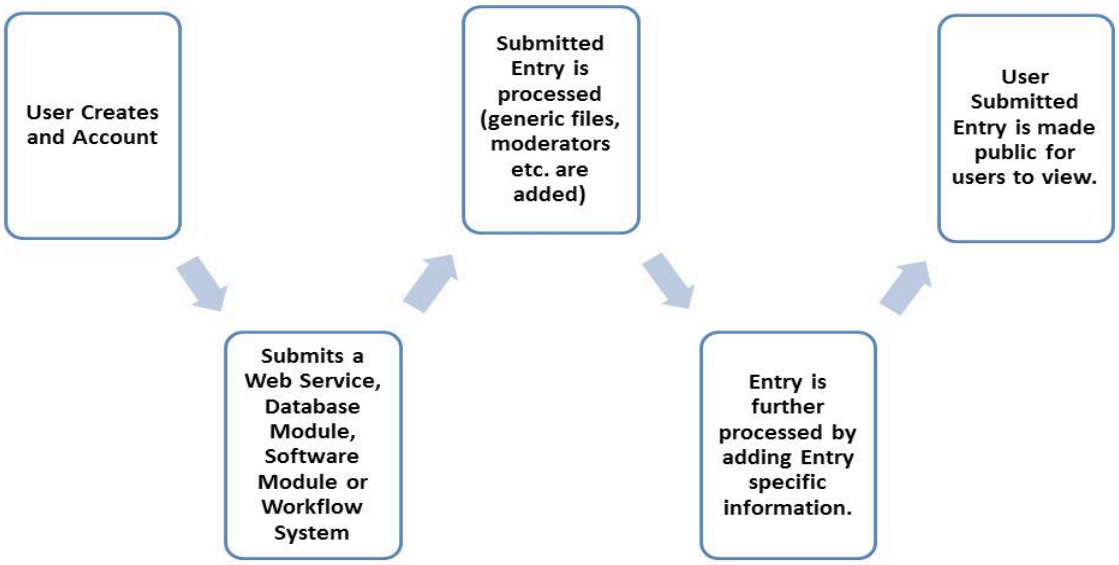

Figure 4: System workflow for new entity submission

Step 1: User creates the account and logs into the portal for authenticity:

Figure 5: Login page

Step 2: User submits a Web service, workflow, database or software module. All entries have a common input page with generic information as listed on Figure 6.

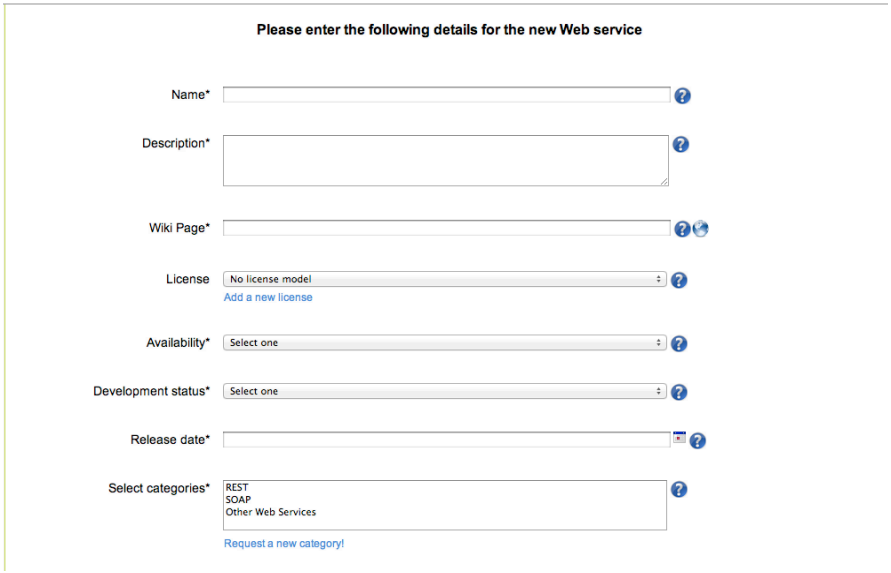

Figure 6: Generic details for entry submission

Step 3: User uploads generic files for certain type of entries like Web service that are processed to prefill fields in the next steps:

Figure 7: Generic page for entry file information

Step 4: Uploaded files are then added to the DB. Then entry specific additional information is manually added or the uploaded files are processed when provided, to automatically fill the entry specific information. Supported information includes:

Web service specific information such as Operations, inputs and outputs;

Workflow specific information such as Inputs and outputs;

Database specific information such as Database size, number of entries, structure;

Software specific information such as Software download, snapshots;

Figure 8 shows the entry specific information page for a database entry.

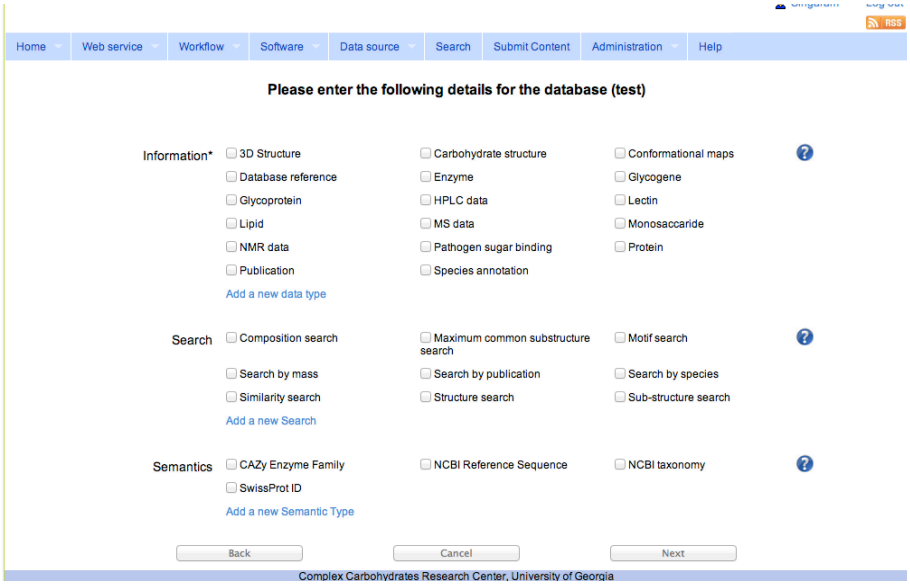

Figure 8: Entry specific information page

Step 5: The uploaded entry can then be modified in the 'Manage Entry' section. These modifications include modifying any content of the entry. 'Manage Entry' also allows adding various other options specific for an entry, and generic options like "make entry public for unregistered users to view", "add publications related to the entry", "add files" that include images, presentations, manuals, release notes etc. related to the entry, tags or keywords for searching entry easily, moderators for the entry, owner of the entry, and wiki page that contains more information about the entry:

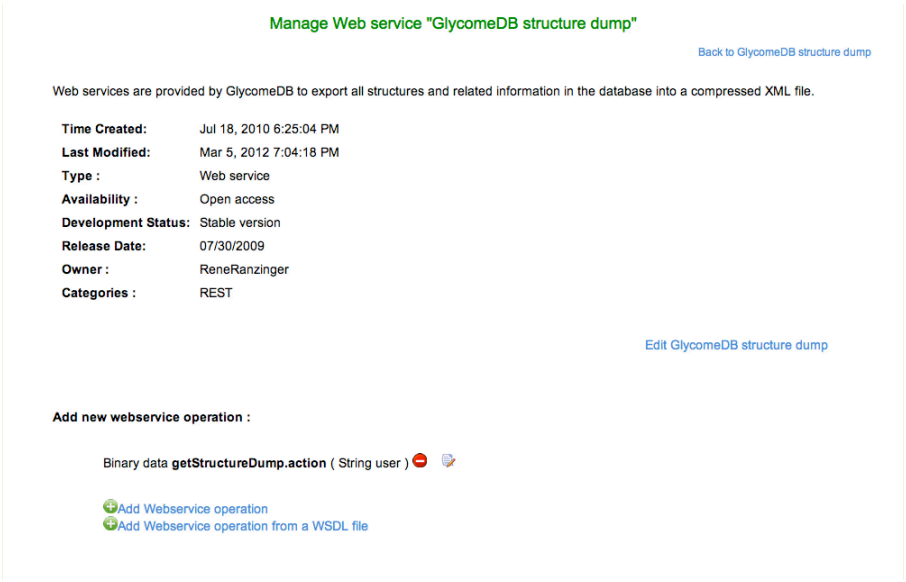

Figure 9: Manage entry page-a

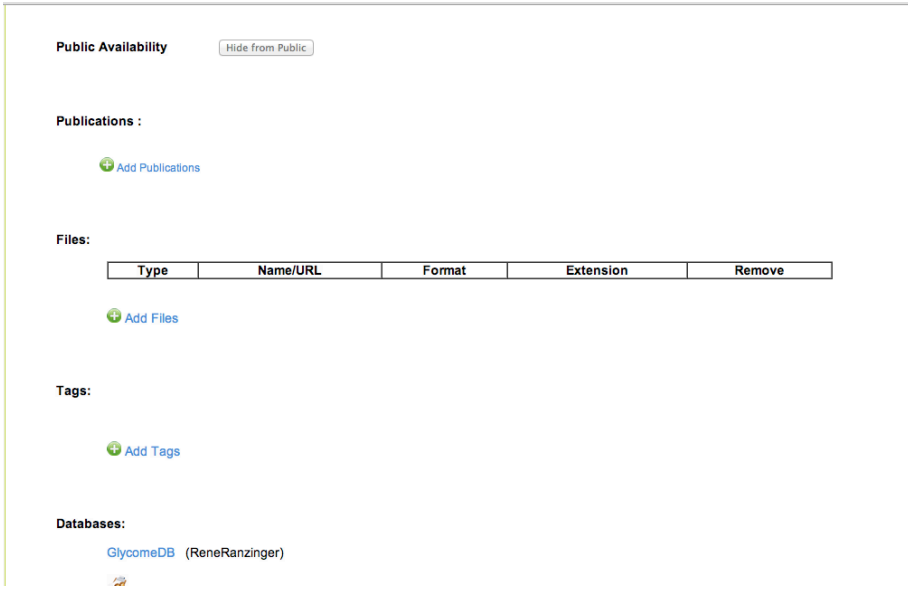

Figure 10: Manage entry page-b

The GlycomicsPortal is a repository where the users can submit and discover Web services, workflows, databases and software modules related to glycomics. The generic client software is integrated with the GlycomicsPortal enabling discovery and invocation making it a one-stop shop.

CHAPTER 4

WEB SERVICES

4.1 Overview

A Web service can be described as a program that performs one or more operations, which is offered by a person or company and located at a remote server [22]. This program is called by the client when the user wants to invoke an operation provided by the Web service, by sending the input data required. The server then returns the result of the operation, which is interpreted by the client. It may be important to note that Web services are designed to be automatically called by software, not by individuals surfing the Web.

4.2 Types of Web services

Web services are broadly classified into two types namely, SOAP and REST.

4.2.1 SOAP

SOAP previously stood for Simple Object Access Protocol [24]. It is a protocol that enables the applications to exchange information over HTTP through SOAP messages. It is an XML based protocol used for communication between the applications for transferring data.

4.2.1.1 Types of SOAP Service

SOAP services are generally classified based on the bindings of the Web service. There are two kinds of WSDL SOAP bindings. It could either be a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. This gives us the following four style/use models [36].

1. RPC/encoded
2. RPC/literal
3. Document/encoded
4. Document/literal

- **RPC Encoded/Literal:**

The main purpose of SOAP RPC is to provide Remote Procedure Calls (RPC) transparently to the client. This means that the client behaves as if it is calling methods on a local object, however, in fact those method calls are translated to XML and sent over the internet, processed on the other end. Once they are processed, the response message is concealed as the return value from the method call.

- **Benefits of using RPC:**

The following are some of the strengths of using RPC:

1. Straightforward WSDL
2. The name of the operation appears in the message. This enables the receiver to easily dispatch the message to the implementation of the operation.
3. RPC/literal is WS-I [10] compliant.

Sample Messages (All example XML code snippets listed in this Chapter are obtained from the reference cited [14])

RPC/Encoded WSDL (1.1)

- Sample Input:

```
<wsdl:message name="DistanceInput">
        <wsdl:part name="p1" type="tns:Point" />
        <wsdl:part name="p2" type="tns:Point" />
</wsdl:message>
```

- Sample Output:

```
<wsdl:message name="DistanceOutput">
        <wsdl:part name="result“ type="xsd:float“/>
</wsdl:message>
```

a) RPC/Encoded SOAP

- Input:

```
<soap:Envelope ..>
 <soap:Body soap:encodingStyle=
     "http://schemas.xmlsoap.org/soap/encoding/">
  <ns:Distance ..>
   <p1 HREF="#id1"/>
   <p2 HREF="#id1"/>
  </ns:Distance>
  <ns:Point id="id1">
   <x>10</x>
   <y>20</y>
  </ns:Point>
 </soap:Body>
</soap:Envelope>
```

- **Document Literal:**
  - Input is any XML document
  - Output, if present, is another one
  - Both are described with XML Schema (in theory, other schema languages are supported)

- **Benefits of using Document Literal:**
  - There is no type encoding information.

- Any XML interpreter can interpret the message as everything in the body of SOAP message is defined in XML schema.
- Document/literal is WS-I compliant, but with restrictions

Sample Messages

a) Document/Literal WSDL (1.1)

- Sample Input:

<rnc:Distance>

distance { p1 { Point }, p2 { Point } }

Point = x {xsd:float}, y {xsd:float}

</rnc:Distance>

<wsdl:message name="DistanceInput">

<wsdl:part name="body" type="rnc:Distance" />

</wsdl:message>

- Sample Output:

<wsdl:message name="DistanceOutput">

<wsdl:part name="result" type="xsd:float" />

</wsdl:message>

b) Document/Literal SOAP

- Sample Input:

<soap:Envelope>

<soap:Body>

<ns:distance>

<p1>

<x>10</x> <y>20</y>

</p1>

```
                <p2>
            <x>100</x> <y>200</y>
                </p2>
        </ns:distance>
            </soap:Body>
    </soap:Envelope>
```

4.2.2 REST:

REST stands for Representational State Transfer [25], in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of stateless operations and arbitrary Web services, in which the service may expose an arbitrary set of operations.

REST is defined in the context of HTTP protocol and also is based on application layer protocol. HTTP uses the following methods [38] and is called based on the requirement,

GET – Requests and retrieves data from the service

POST – Submits data to the service to be processed

PUT – Uploads a representation of the specified resource

DELETE – Deletes the specified resource

4.2.2.1 RESTful Services:

A REST service that conforms to the following REST constraints is called RESTful service [26]. The constraints are:

- Client-Server – Follows client-server architecture.

- Stateless – client-server communication must be stateless, in which each request from the client contains all required information.
- Cacheable – The response should be defined as cacheable or non-cacheable.
- Layered system – The architecture can comprise of multiple layers.
- Code on demand – Client functionality can be extended by downloading and executing the code.
- Uniform interface – implementations are decoupled from the services they provide and emphasizes uniform interface between components.

4.3 Technologies in Web services:

- **WSDL**

WSDL stands for Web service Description Language [12]. It is an XML based language and is used to provide the description of services available. The WSDL file includes the details about the Web service including how the service can be called, what parameters are accepted, what operations it performs etc. WSDL is often used in combination with SOAP and XML Schema to provide Web services over the Internet.

A WSDL definition is divided into separate sections. These sections specify the details about the logical interface and physical Web service. The details include the HTTP port number, details about how the SOAP payload is represented and which transport protocol is used. WSDL is discussed in greater detail in chapter 5.

- **WADL**

WADL stands for Web Application Description Language [8]. WADL is lightweight, easier to write and easier to use than WSDL. However, it may not be as flexible as WSDL. It is sufficient for any REST service and much less verbose. A large number of web-based enterprises provide HTTP-based applications that provide access to their internal data. Usually these applications are designed using textual documentation that is sometimes supplemented with more formal specifications such as XML schema for XML-based data formats. "WADL is designed to provide a machine process-able description of such HTTP-based Web applications [8]"

- **UDDI Open Standards**

UDDI stands for Universal Description, Discovery and Integration [9]. UDDI is used for listing the services that are available. UDDI makes discovery of services possible.

4.4 Web service data flow:

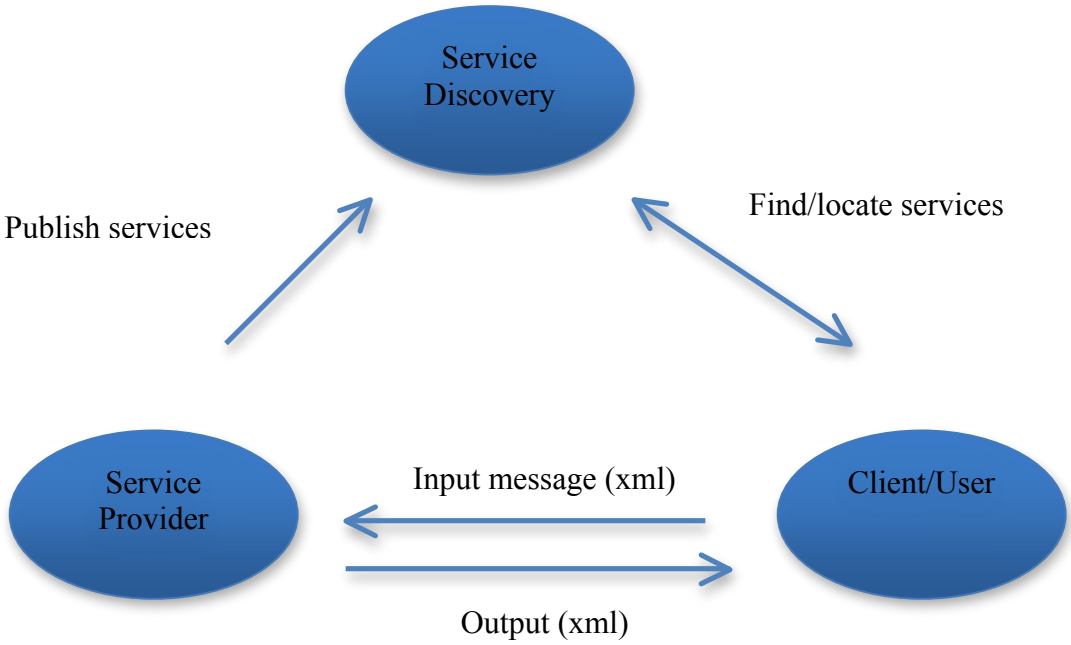

Figure 11: Web service data flow Service-oriented Architecture (SOA) [28]

CHAPTER 5

WSDL

5.1 WSDL Overview:

Once the Web service is developed, the Web service is published with its description and a link to UDDI (Universal Description, Discovery and Integration) repository. This enables the potential users to easily find and use the Web service. Potential users request the Web service's WSDL file to find the location, function calls and access to the function calls etc. [11]. Then they use this information in WSDL file to form a SOAP (Simple Object Access Protocol) request.

A WSDL definition consists of the following main parts:

- A description of the messages that can be passed
- The semantics of passing the message
- Specified encoding information
- The Endpoint information

5.2 Types of WSDL:

The earlier version was WSDL 1.1 and the current version is WSDL 2.0. WSDL 2.0 came with several modifications to increase the interoperability; however, very few vendors support WSDL 2.0 today due to the same interoperability issues [29].

Here are the main underlining differences between WSDL 1.1 and WSDL 2.0

WSDL 1.1	**WSDL 2.0**
<definitions>	<description>
<portType>	<interface>
<port>	<endpoint>
<message>	<message> is removed and defined inside <operation>

5.3 Description of a WSDL file:

The following example shows the WSDL definition of a simple service providing stock quotes. Here is a sample WSDL file; this example uses the SOAP encoding [12]:

```
<?xml version="1.0"?>
<definitions name="StockQuote"
        targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
<message name="GetTradePriceInput">
   <part name="tickerSymbol" element="xsd:string"/>
   <part name="time" element="xsd:timeInstant"/>
</message>
<message name="GetTradePriceOutput">
   <part name="result" type="xsd:float"/>
</message>

<portType name="StockQuotePortType">
   <operation name="GetTradePrice">
      <input message="tns:GetTradePriceInput"/>
      <output message="tns:GetTradePriceOutput"/>
   </operation>
</portType>

<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="GetTradePrice">
      <soap:operation soapAction="http://example.com/GetTradePrice"/>
      <input>
         <soap:body use="encoded" namespace="http://example.com/stockquote"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
```

```
        <output>
            <soap:body use="encoded" namespace="http://example.com/stockquote"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>>
   </binding>

   <service name="StockQuoteService">
      <documentation>My first service</documentation>
      <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote"/>
      </port>
   </service>
</definitions>
```

This sample code was taken from [12].

5.4 Entities of a WSDL file:

A WSDL file consists of the following entities:

1. Types
2. Message
3. Operation
4. Port Type
5. Binding
6. Port
7. Service

- **Types**

A container for data type definitions using some type system (such as XSD). The “types” element encloses data type definitions that are relevant for the exchanged messages. WSDL prefers the use of XSD as the canonical type system, in order to increase interoperability and platform neutrality.

- **Message**

An abstract type definition of the data being communicated, which can be input, output or fault. Messages can be composed of one or more logical parts. Each logical part associated with a type from some type system using a message-typing attribute. There are several message-type attributes currently defined in WSDL

  - **element**. Refers to an XSD element using a QName.
  - **type**. Refers to an XSD simpleType or complexType using a QName.

- **Operation**

An abstract description of an action supported by the Web service. The operation contains the input and output messages related to it. The message can have the list of elements definition directly under it or in the WSDL types based on the type of WSDL.

- **Port Type**

Port Type is an abstract set of operations as well as abstract set of messages supported by one or more endpoints. The port type name attribute provides a unique name among all port types defined within the enclosing WSDL document.

- **Binding**

A binding is used to define the message format and protocol details for operations and messages defined by a portType.

- **Port**

A port is defined as a single endpoint defined as a combination of a binding and a network address.

- **Service**

A service is defined as the collection of related endpoints. A service groups a set of related ports together.

CHAPTER 6

IMPLEMENTATION OF WEB SERVICES INVOCATION AND AUTOMATIC POPULATION OF WEB SERVICES

6.1 Overview:

This chapter describes the concepts, technologies, design and implementation of the Web service invocation system and the automatic Web service population system. The Web service invocation system comprises of the Generic SOAP and REST clients.

6.2 Technologies:

- Apache AXIS 2 framework [30]

Apache Axis 2 is the SOAP and WSDL engine that supports the Web service invocation system providing required functionalities. It is a Java-based implementation of both client and server sides of the Web service system. The Generic SOAP client implements the Axis 2 framework for sending SOAP messages, receiving and processing SOAP messages which constitutes the Web service invocation system. Apache Axis 2 provides an API with functions that enables us to make use of its features.

Figure 12 below illustrates the integration of Apache Axis 2 in the GlycomicsPortal and Generic SOAP client. The User application in the Figure 12 denotes the Web portal and the SOAP client that uses the Axis 2 Client API. The constructed SOAP message is sent from the SOAP client to the Axis 2 framework where it is handled; this is denoted by Handler in Figure 12 and transported over the Internet through SOAP protocol. It is then received at the server end, handled and the Web service business logic performs the defined operation and sends the

message back to the client. The SOAP client at the client side handles the received message.

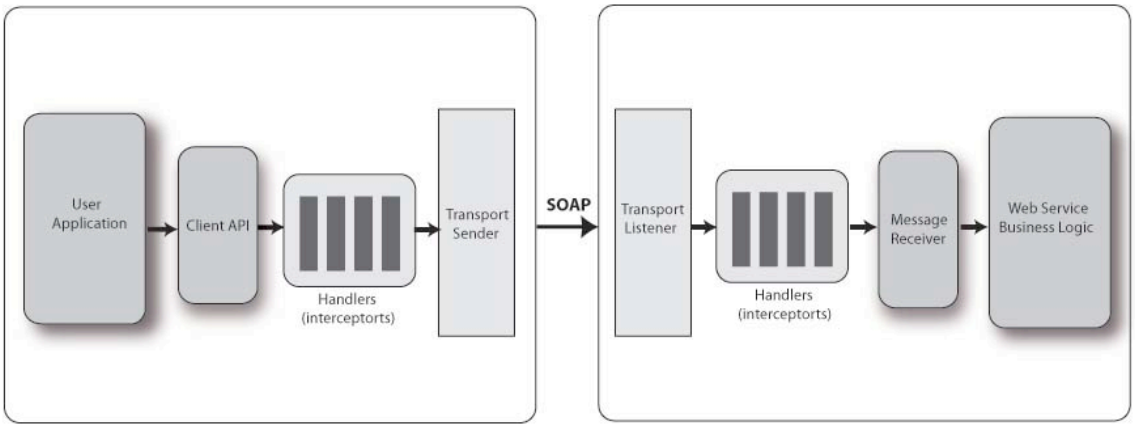

Figure 12: Apache AXIS 2 framework [31]

- WSDL4J API

WSDL4J stands for Web service Description Language for Java [32]. It is a toolkit for WSDL support in Java, which enables representation and manipulation of WSDL documents. WSDL4J API is used in the GlycomicsPortal and Generic SOAP Client to parse and manipulate the WSDL files. The WSDL4J can parse the WSDL structure that includes the service, bindings, port types, operations and messages but not the part of WSDL documents that contains the XML schema definition. That is where the JDOM parser is used in addition to WSDL4J to support manipulating WSDL files.

- JDOM API

JDOM is an API for Java representation of XML documents [33]. JDOM provides the ability to read, manipulate and write XML documents using Java. It is used in the GlycomicsPortal and the Generic SOAP client to create and interpret XML message in the WSDL document and received/sent SOAP messages.

6.3 Automatic Population of Web services:

This system has two major roles in the GlycomicsPortal.

- Parse the WSDL to prefill the generic details about a Web service during registration process
- Automatically register operations provided by a SOAP service with the portal by parsing the WSDL

The user is allowed to upload or provide the location of the WSDL definition for the Web service during registration process as demonstrated in Figure 7. The uploaded or located WSDL document is then parsed using WSDL4J parser to provide the list of ports to register and then prefill the details of Web service specific information like service provider and URL based on the chosen port during registration.

The automatic population of Operations in the Web service is implemented by parsing the WSDL document using the WSDL4J parser and providing the list of operations. Then the details of the chosen operation that includes the number of inputs, outputs and faults are prefilled from the WSDL. The user further proceeds with the registration process where the input parameters, output parameters and fault messages are prefilled with each parameter's Name and Type from the WSDL. The steps in the automatic population of operations in a registered Web service are as follows,

- Display list of operations
- Enter operation details
- Enter input parameters
- Enter output parameters
- Enter fault message details

6.4 SOAP Invocation Manager:

A registered Web service can be invoked by navigating to the Web service entry page in the portal. The user clicks on 'Invoke operation' link at a Web service entry page that exists next to an operation provided by the Web service. This calls an event handler that sends the URL of the Web service, which is the location of WSDL file that is stored either in a remote location or in the portal server and the operation information to the SOAP Invocation Manager.

SOAP Invocation Manager receives the following required information

- Location of the WSDL file
- Operation name

It checks for the type of the SOAP service or WSDL, which can be basically rpc-encoded or doc-literal. It routes the invocation process to proceed based on the type of WSDL.

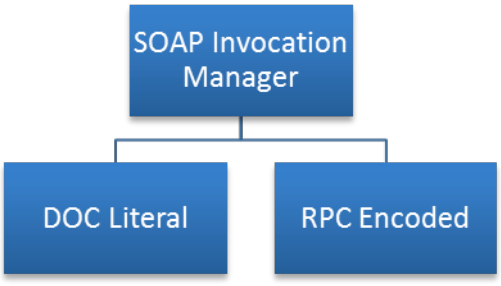

Figure 13: Soap Invocation Manager

6.4.1 DOC-Literal

If the type of the WSDL is in doc-literal format, then the SOAP Invocation Manager performs the following operations:

(1) Get the input message for the current operation from the WSDL.
(2) Locate the input message definition in the WSDL document.
(3) Locate the part element defined in the input message. The part element is located in the <wsdl:types> which is defined in the XML schema definition of the WSDL. The element located would have a list of simple and complex elements under it. It then retrieves the list of elements or its children one by one and routed as follows:
    a. If the element is a simple element then it reads the detail of each element which include:
        i. Name – Name of the element
        ii. Type – Type of the element
        iii. Value – Value that is empty and has to be entered by the user
        iv. Description – Description or annotation or documentation specified about the element
        v. Required – Specifies if the element is mandatory
    b. If the element is a complex element then it recursively calls the same function that retrieves the list of children. That function retrieves the list of the current complex element's children. This returns the simple elements under this complex element and in case of any further complex elements the recursive call continues until the leaf nodes are reached.

6.4.2 RPC-Encoded:

If the type of the WSDL is in rpc-encoded format, then the SOAP Invocation Manager performs the following operations,

(1) Get the input message for the current operation from the WSDL.
(2) Locate the input message definition in the WSDL document.
(3) Retrieve the list of parts under the input message. The major difference between rpc-encoded and doc-literal WSDL comes into play at this point, where the list of elements at the first level are defined directly under the input message instead of the XML schema definition <wsdl:types>.

Now, based on the type of part element, we proceed as follows,

a. If the part is an element of simple type then it reads the following detail of each element:
    i. Name – Name of the element
    ii. Type – Type of the element
    iii. Value – Value that is empty and has to be entered by the user
    iv. Description – Description or annotation or documentation specified about the element
    v. Required – Specifies if the element is mandatory

b. If the part element is of complex type then, it locates the element in the <wsdl:types> in XML schema definition. After locating the complex element, the lists of its child elements are read. If any complex elements are found then, it recursively calls the same function that retrieves the list of the current element's children. This returns the list of all elements under an operation until the leaf nodes are reached.

6.4.3 Input Element List

All the read elements are used for populating the input element list that is built dynamically for both rpc-encoded and doc-literal WSDL. By the end of traversing through the WSDL tree structure, we have a completely filled input element list with the list of all the elements under the invoked operation with details. It is possible to trace the tree back from this input list as it contains all the information required to rebuild the WSDL structure.

After populating the input element list, it is displayed to the user with all the available information including Name, Type, Required and Description for every input parameter. Each element or input parameter displayed to the user has an empty field next to it where the users can enter their input values.

Once the values are entered by the user, we fill up the value attribute that specifies the user's input value for an element by traversing and repopulating the input element list. After creating a completely filled list with all the user values, we call the Generic SOAP client by providing the WSDL URL, operation name and the filled input element list.

6.5 SOAP Client

The generic SOAP client is the independent module that gets the required information from the portal and invokes the SOAP Web service. The functionality of the SOAP client is as follows:

- Prepare for invocation by gathering the required information from the GlycomicsPortal
- Construct the SOAP message based on WSDL
- Invoke the Web service
- Get the XML response back and send it to the result handling system

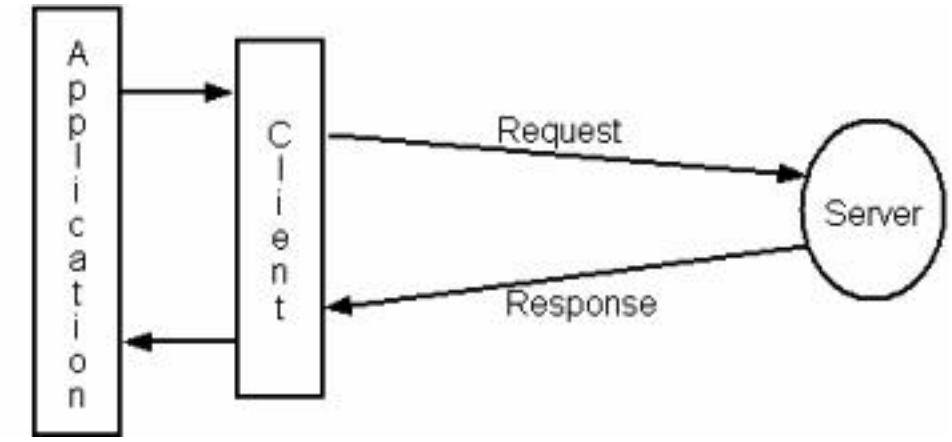

Figure 14: SOAP Client Architecture [37]

The SOAP client uses the Apache Axis 2 framework to invoke the SOAP Web service by passing the created payload (constructed message) and operation qname. It is

invoked in a synchronous and blocking manner. Figure 14 illustrates the synchronous functionality in which the client sends out a request and waits for the server to respond. A dynamic service client is created with every given SOAP service for invocation.

6.5.1 Service client

The Service client API is provided by the Apache Axis 2 framework to invoke the Web service. This acts as the mediator between the client and the actual service on the web. The functions from Service client API are used for interacting (sending/receiving data) with the actual Web service. The Service client can be used to invoke the Web service in a synchronous or asynchronous manner.

An Axis 2 runtime called Configuration Context is required to invoke the Web service. Configuration Context consists of configurations, parameters, and other relevant data. Semantically, ConfigurationContext is similar to ServletContext in an application server. Configuration context can be specified explicitly or used by default settings when no special arguments are required to call a service. Service client can be either static or dynamic.

- Static service client

A static service client is used for calling a specific service and may not be an ideal choice in our system. It requires the options object to be set with the service end point reference and operation details.

- Dynamic Service Client

The idea of a dynamic client is to create a ServiceClient on the fly, or simply create a client for a given WSDL at runtime and use the created ServiceClient instance to invoke the corresponding service (the service corresponding to the WSDL). When we create the ServiceClient in this manner, it will create and configure a service proxy inside the client. The constructor for creating a dynamic client is as follows:

```
ServiceClient dynamicClient = new ServiceClient(   configContext, wsdlURL,
                                                   wsdlServiceName, portName );
```

- configContext – This system uses null for this parameter as it uses the Axis 2 default configuration context instead of defining an explicit configuration for a client

- wsldURL**:** This argument specifies the URL for the WSDL file and is filled with the URL from the SOAP Invocation Manager.

- wsdlServiceName**:** A WSDL document might have multiple service elements, the QName of that service element can be passed as an argument. We do not want to pick a specific service element, so this system uses null for this argument which uses the default first service element.

- portName**:** A service element in a WSDL file could have multiple ports. If we opt to select a specific port, then the name of the port is passed as a value for this argument. Since the ports are handled and filtered while registering the service with the portal through automatic Web service registration, the ports need not be explicitly defined here, so no option is specified for the port and the default port is selected.

- Creating Payload

Create payload is the process of constructing the SOAP message body by specifying the input parameter names and values. It is implemented using the Axis 2 API function OMElement.

6.5.2 OMElement

OMElement is the SOAP message container, which builds the message dynamically reading information from the WSDL. The SOAP message construction

process is handled by the OMElement Handler. The invoked operation information and other required WSDL details are passed to the OMElement Handler to construct the SOAP message. For example OMElement for a single element will look like,

<programming>java</programming>

In the above OMElement, programming is the parent element and Java is the text or value of that element.

- OMElement Handler:
  OMElement Handler does the following operations to process and construct the SOAP message from the WSDL and available user input data.
  - Create an OMElement with the operation information. This will be the SOAP message header that helps in dispatching the SOAP request.
  - Parse the WSDL document to get the complete tree structure of elements under the given operation.
  - Create an OMElement for every processed element maintaining its parent-child relationship as defined in the WSDL structure.

The OMElement Handler is a recursive function that parses the WSDL tree structure to dynamically construct the SOAP message. The code snippet on page 44 explains the working of the OMElement Handler process. The root element of the WSDL, operation information and XML schema information are the parameters required to start the handler process.

After the handler is initiated, it parses the WSDL to find the list of root element's children, which would be defined in the XML schema definition of the WSDL. While traversing through the list of children, if the element is of a simple type, a child OMElement is created and added as a child of the root OMElement in the SOAP message. If the element encountered is of complex type or a list, OMElement Handler process is called recursively passing the complex element as the root element. The same process is recursively called until the leaf nodes of the WSDL tree structure are reached.

OMElement Handler Recursion process (Code snippet for doc-literal recursion process) is as follows:

```
If (root element) then
        Create root OMElement object
For list of children do
{
        if (element is primitive type) then
                create child OMElement object and
                add as child to root OMElement
        else if (element is of complex type) then
                call OMElement Handler passing the complex element as root element
        else if (element is a list) then
                call OMElement Handler passing the list element as root element
}
end
```

The OMElement object is completely filled by the end of recursion and contains the final SOAP message to be sent over. The SOAP client then invokes the Web service with the SOAP message as follows,

```
serviceClient.sendReceive(operation qName, OMElement);
```

operation qName – refers to the WSDL representation of the invoked operation
OMElement – refers to the complete SOAP message.

Doc-Literal and RPC-Encoded WSDLs are handled in different ways for contructions of the SOAP message because of the difference in WSDL structure. Doc-Literal follows the same steps as mentioned above for SOAP message construction. RPC-Encoded has the following process:

- RPC OMElement Handler

RPC OMElement Handler works in a different way compared to the doc-literal because of the WSDL structure. The root element of the WSDL, operation information, input message and XML schema information are the parameters required to start the handler process.

After the handler is initiated, it parses the WSDL to find the list of part elements under the operation's input message. This list of part elements is the operations first level of child nodes. While traversing through the list of children, if the element is of a primitive data type, a child OMElement is created and added as a child of the root OMElement in SOAP message. If the element encountered is of complex type or a list, OMElement Handler process is called passing the complex element as the root element. The same process is recursively called until the leaf nodes of the WSDL tree structure are reached. The following code snippet explains the working of the RPC OMElement Handler process.

```
If (root element) then
        Create root OMElement object
For list of parts under input message do
{
        if (part element is simple type) then
                create child OMElement object and
                add as child to root OMElement
        else if (element is of complex type) then
                call OMElement Handler passing the complex element as root element
}
```

- SOAP Request

The SOAP message contains the information required for invoking the operation provided by the SOAP service. SOAP message can be split into two parts, i.e., the SOAP envelope and the SOAP body. The SOAP body is created dynamically based on the WSDL by the SOAP client. The SOAP envelope refers to the wrapper around the SOAP body making a complete SOAP message that can be transmitted over the Internet to the SOAP service. The SOAP envelope is created dynamically by the Axis 2 framework when an invocation call is made according to the configuration context.

- SOAP Response

The SOAP Web service when invoked by passing the constructed SOAP message, gives back an XML response that contains the result. The XML response received from the SOAP service is sent to the Result Handling system to process and display the result to the user.

6.6 REST Client:

The REST services registered with the GlycomicsPortal can be invoked using the REST client. The portal contains information about the REST service URL and the input parameters required to invoke it. It is implemented by constructing the service URL including all the parameters and then making an HTTP request. The REST invocation is handled and processed by two systems,

- REST Invocation Manager
- REST Client

- REST Invocation Manager

The REST Invocation Manager is responsible for communicating with the Portal to retrieve REST Web service related information and construct the URL. The steps in REST Invocation Manager are as follows:

(1) Retrieve the complete details about the REST service that user wants to invoke

(2) Locate the specific operation to be invoked by the user in that REST service

(3) Read the URL and input parameters for the specific operation in the given REST service from the portal database

(4) Display the input parameters with information including Name, Description and Type with a blank text field next to it to allow the user to enter inputs in a web page

(5) Get the inputs for the displayed parameters from the user

(6) Construct the REST invocation HTTP call as follows:

- The first part of the HTTP call will be the URL of the REST service
- The operation name is appended to the base URL followed with a "?".
- Each input parameter and its values from the user are then appended to the URL separated by "&" according to the HTTP invocation standards.

(7) Transfer the control to the REST client by passing the constructed URL.

For example, a constructed REST HTTP call will look like,
http://www.glycome-db.org/database/getStructureEncoding.action?id=1&type=GlydeII

In the above example, "http://www.glycome-db.org/database/" is the REST service URL. "getStructureEncoding.action" is the operation provided by the Web service. "id" and "type" are the two input parameters with values "1" and "GlydeII", respectively.

- REST Client

REST invocation system comprises of the REST client that takes in a complete URL and invokes the Web service.

- HTTP Request

Java provides a HTTP connection API that is used to send a request to the given REST service URL. Our system supports the GET method for REST invocation. An HTTP connection for a given URL is opened as follows:

```
HttpURLConnection conn = url.openConnection();
```

The steps in the REST client for invoking a service is as follows:

(1) Get the complete REST service URL from the REST Invocation Manager

(2) Invoke the REST service using an HTTP request

(3) Listen to the HTTP open connection for stream of data as a response

(4) Call the Result Handling System to parse the results and display it to the user

6.7 Result Handler:

The Result Handling system is used by both SOAP and REST client to process the XML response received.

- XML Parser

The XML parser is responsible for parsing and returning only the content to the user ignoring all tags and other schema information. It implements a recursive function that traverses through the XML file from root till the last child node. It prints the content, which is the text in each tag as it traverses and forms a JSON output that is readable.

- Other Formats

Web services can return responses in formats other than XML. It can be an image or a file. When the format is not XML or text, the result handling system interprets those formats and prints them on the browser directly.

- SOAP Client result handling

The SOAP service returns an OMElement object which is interpreted by parsing through an XML parser to get the content inside the tags forming the result. The retrieved content from the XML parser is displayed to the user as the result of the invoking the service.

- REST Client result handling

The REST client returns the stream of data as a response from the service to the Result handling system. If the stream of data received is of type text or XML, then it is sent to the XML parser for further processing and retrieving required content. If the data is of different type, then it is interpreted and displayed to the user.

6.8 Comparison of Web service invocation between other systems and GlycomicsPortal:

The service that we are invoking for comparison is NCBI Blast SOAP service and the operation called 'run' provided by NCBI Blast (http://www.ebi.ac.uk/Tools/webservices/services/sss/ncbi_blast_soap). It is registered in EBI and GlycomicsPortal. The first step for invoking a Web service is to locate the Web service in the repository. The NCBI Blast SOAP service is located in EBI as shown in Figure 15. The operations in the NCBI Blast service are shown in Figure 16. The two pages in figure 15 and figure 16 provide all the information required to invoke the NCBI Blast Web service.

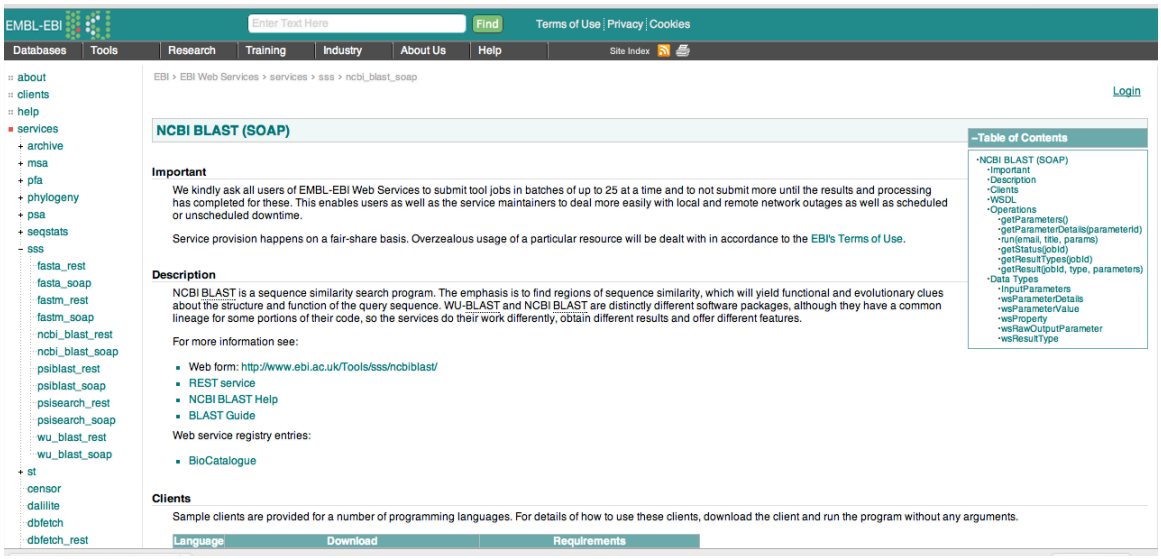

Figure 15: EBI- NCBI Blast Web service page

(http://www.ebi.ac.uk/Tools/webservices/services/sss/ncbi_blast_soap)

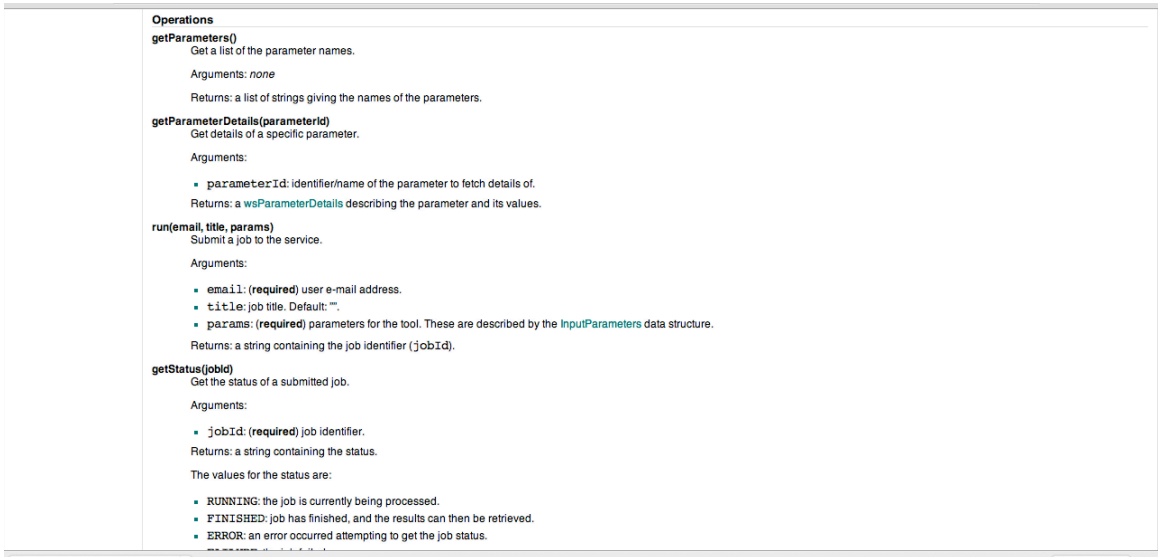

Figure 16: EBI- NCBI Blast Web service operations

(http://www.ebi.ac.uk/Tools/webservices/services/sss/ncbi_blast_soap)

The next step for invocation of the Web service is to copy the gathered information that includes WSDL location and operation name from EBI about NCBI Blast to SOAPUI, which is a generic SOAP client. Figure 17 represents the interface for SOAPUI application and illustrates the invocation of NCBI Blast service. The left part of Figure 17 represents the SOAP request, which is generated, based on the WSDL file and chosen operation. It is the xml SOAP message that contains the set of inputs for "run" operation as tags and a "?" in between the opening and closing tag, which should be replaced with the actual input by the user.

For instance: <email>?</email> represents the input parameter "email" and the "?" represents the user input value. For example, the tag after replacing "?" with user input will be, <email>abc@abc.com</email>. In a similar way, all the tag's content in the SOAP message should be modified by the user with input values. The "run" button is then clicked which invokes the "run" operation in NCBI Blast service and returns the SOAP response as an xml message represented on the right side of figure 27. The SOAP response contains information about the "job id" which is the result of "run" operation.

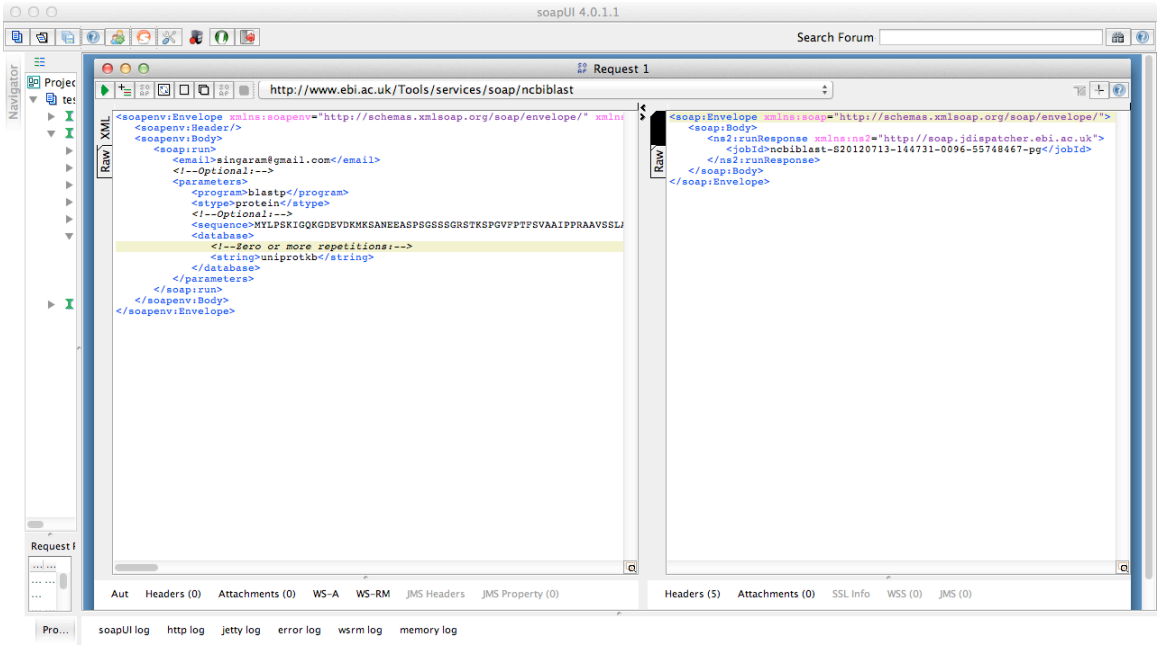

Figure 17: SOAPUI – Invocation of NCBI Blast

In comparison to invoking the NCBI Blast service using EBI and SOAPUI, let's invoke the same with GlycomicsPortal. I have already discussed about navigating to a Web service entry in the portal and viewing its information. Figure 18 represents the entry page for NCBI Blast service registered with the GlycomicsPortal. It provides information relevant to the Web service. There is an 'Invoke Operation' link next to every operation listed out in the Web service entry page for all SOAP and REST services. In comparison to EBI and SOAPUI, where the user has to manually copy all the required information from EBI repository to the SOAPUI interface, in the GlycomicsPortal the user just clicks on the 'Invoke Operation' link next to the operation to invoke the Web service.

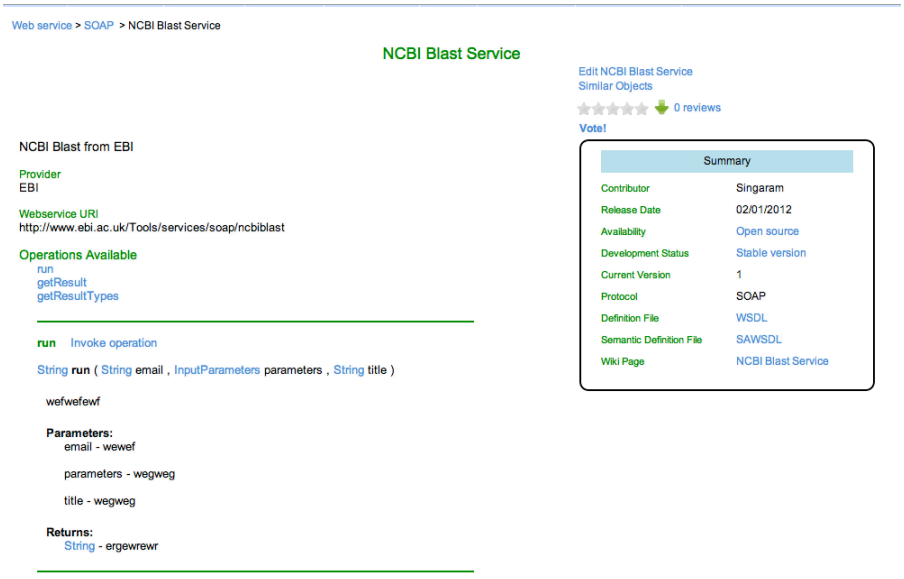

Figure 18 – NCBI Blast service in GlycomicsPortal

When the 'Invoke Operation' link next to "run" operation is clicked, it processes the implementation of SOAP client as discussed in this chapter and displays the Web service invocation page (Figure 19). The Web service invocation page displays the input parameters for the invoked "run" operation with the following information,

- Name
- Type
- Description

The information above are retrieved from the WSDL file and displayed intuitively in a user-friendly interface compared to the XML SOAP message displayed by SOAPUI which has to be modified by the user.

Figure 19 represents the Web service invocation page that is displayed with the input parameters and its details to the user. It contains a blank field for every input where the user enters the input value.

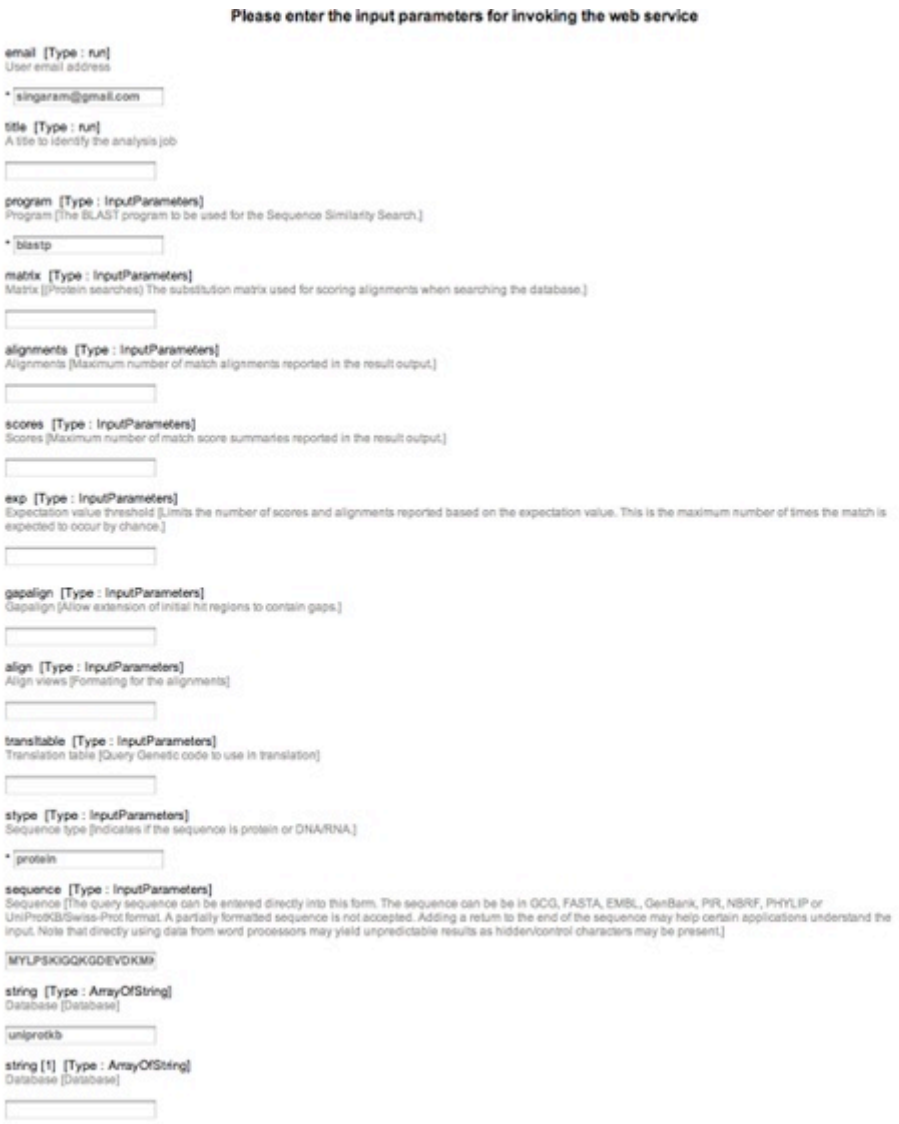

Figure 19: Web service invocation page

The user enters the input for the parameters listed and clicks on “Run” which initiates the generic SOAP client process and then the result handling system as discussed

in this chapter. The Result page is displayed after invoking the operation and providing the required inputs. Figure 20 represents the result page for invoking "run" operation from NCBI Blast service. The result contains the "job id" that is parsed from the SOAP response and displayed to the user compared to SOAPUI that displays the entire SOAP response from which the user interprets the result.

Figure 20: Results page for run operation

The invocation of "run" operation can be further continued by invoking "getResult" operation provided by the NCBI Blast service that takes in the result of "run" operation which is "job id" and type of result as input and which returns the result of the job. Figure 21 shows the Web service invocation page displaying the input parameters for "getResult" operation. When this is invoked by clicking on "Run", it returns the result (Figure 22) that displays the result of submitted job.

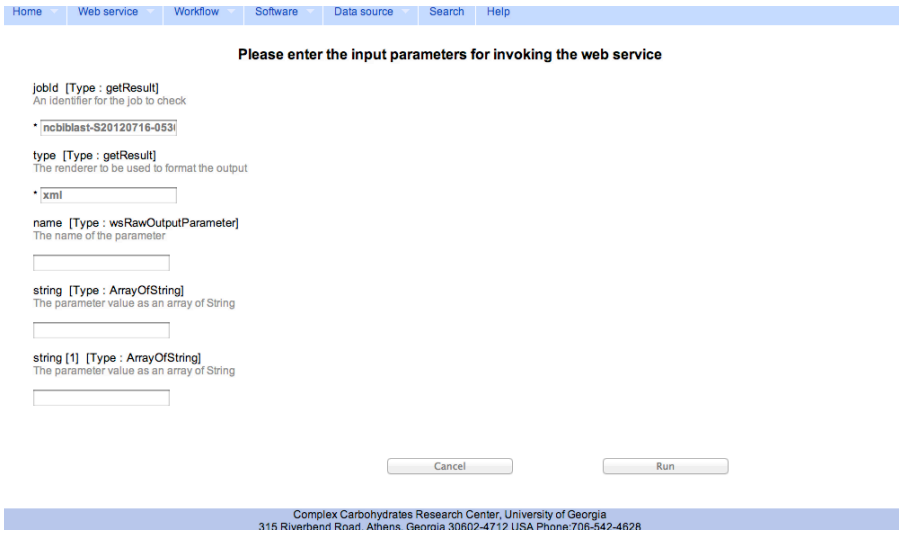

Figure 21: Input page for getResult operation

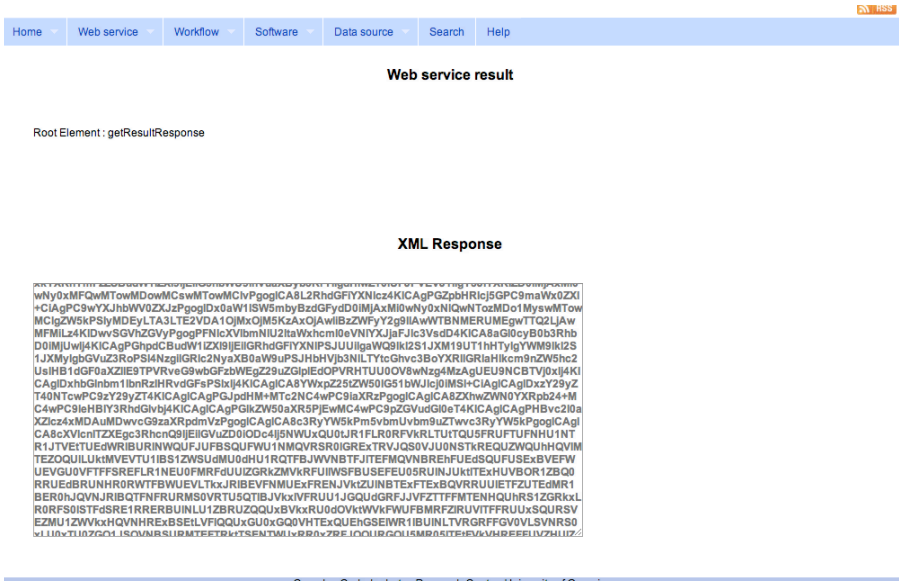

Figure 22: Output page for getResult operation

- REST Invocation

The REST Web services when invoked from portal displays an input page similar to the SOAP services. It lists out all the input parameters for the invoked operation, which are processed from the portal as explained in chapter 6. Figure 23 shows the input page displayed when invoking the Glycome DB REST service registered with the portal.

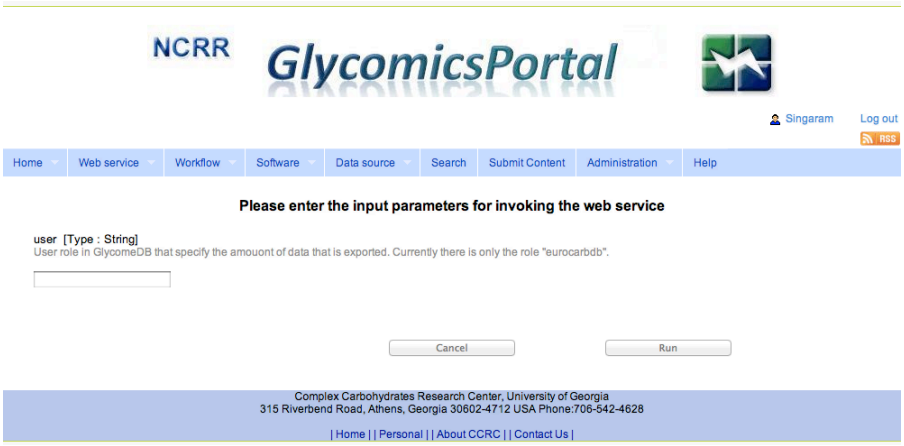

Figure 23: Rest service input page

The user enters the input values for the displayed fields and clicks on "Next" which processes the invocation using REST client and result handler as explained in this chapter. Figure 24 represents the output received from the Glycome DB REST service.

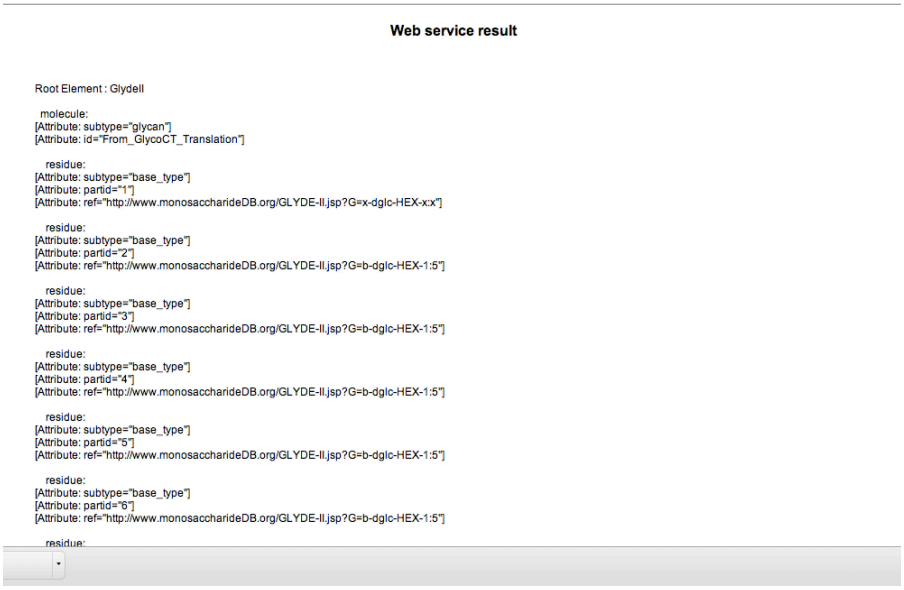

Figure 24 – REST service result page

CHAPTER 7

CONCLUSION AND FUTURE WORKS

Currently there is no widely available system that is a combination of Web service repository and Generic SOAP and REST service client. This thesis has achieved bringing the Web service repository (GlycomicsPortal) and the Web service Invocation system (SOAP and REST Generic Client) together to create a Web service Discovery and Invocation tool with Automatic Web service population.

Thus, the generic client software allows the user to invoke the full functionality of diverse Web services registered with the GlycomicsPortal and saves a considerable amount of effort by individuals who merely want to evaluate or use the service. The table below explains the operations and services currently supported by our system:

Functionality	**Supported Context**	**Unsupported Context**
SOAP Client	Doc-literal and rpc encoded	Doc-encoded and rpc-literal
REST Client	GET Method	PUT, POST, DELETE Methods
RESTful Client	GET Method	PUT, POST, DELETE Methods
Automatic Population of Portal DB with operations defined in a WSDL for a registered Web service	Doc-literal (simple types)	Rpc-encoded simple and complex types doc-literal complex types
Output Handling	XML, Text	Binary (e.g., base64, images)

7.1 Future Work

Future works in system includes adding support for doc-encoded and rpc-literal types for the SOAP client. Even though they are not commonly used this could make the system support wide range of services. To support this the OMElement handler in the system needs to be modified to support different structure of the two types of WSDLs. When the structure of the WSDL until the XML schema definition is handled using a new handler, then the existing OMElement Handler can be reused for the complex elements in the WSDL.

Adding PUT, POST and DELETE methods for the REST Client is also another future work to be considered. To support this, the existing REST Invocation manager needs to be modified from constructing the REST URL to construct the payload at the REST client and reuse the existing result handling system.

Also, automatic population for rpc-encoded and doc-literal complex type needs to be added. To support this feature, the automatic population of web service operations should be modified. There are parsers that already exist to retrieve the information about doc-literal complex elements and rpc-encoded WSDL elements. These retrieved elements can be used to prefill the registration process of operation, its input / output parameters and fault messages, which will enable the system to automatically populate operations from doc-literal and rpc-encoded WSDLs.

REFERENCES

1. About Bio Catalogue. BioCatalogue. "http://www.biocatalogue.org/". Retrieved on May 11, 2012.

2. Our Mission Areas. European Bioinformatics Institute. "http://www.ebi.ac.uk/". Retrieved on May 21, 2012.

3. What is soapUI. soapUI. "http://www.soapui.org/.". Retrieved on May 10, 2012.

4. Generic Soap Client, SoapClient.com. " www.soapclient.com/". Retrieved on May 22, 2012.

5. Xiaoying Bai, Wenli Dong, Wei-Tek Tsai, Yinong Chen. "WSDL-Based Automatic Test Case Generation for Web services Testing". Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering, 2005.

6. Philipp Leitner, Florian Rosenberg, Schahram Dustdar. "Daios: Efficient Dynamic Web service Invocation". IEEE Computer Society, IEEE Internet Computing. 2009.

7. Matthew J. Duftler, Nirmal K. Mukhi, Aleksander Slominski and Sanjiva Weerawarana. "Web services Invocation Framework (WSIF)". IBM T.J. Watson Research Center. 2011

8. Web Application Description Language. W3c.org. "http://www.w3.org/Submission/wadl/". Retrieved on May 1, 2012

9. What is UDDI. W3schools.com. "http://www.w3schools.com/wsdl/wsdl_uddi.asp". Retrieved on May 1, 2012

10. About WS-I. Web services Interoperability Organization. "http://www.ws-i.org/". Retrieved on April 1, 2012

11. Overview of WSDL, Oracle Sun Developer Network. "http://developers.sun.com/appserver/reference/techart/overview_wsdl.html" Retrieved on May25, 2012

12. Web service Description Language. W3c.Org. "http://www.w3.org/TR/wsdl". Retrieved on May 1, 2012

13. Generating deploy code. WebSphere Studio. "http://publib.boulder.ibm.com/infocenter/adiehelp/v5r1m1/index.jsp?topic=%2Fcom.ibm.etools.prodovr.wsinted.doc%2Fhtml%2Fcgendply.html". Retrieved on May 1, 2012.

14. Dan Gunter, "Document Literal vs. RPC Encoded SOAP". Lawrence Berkeley National Laboratory. 2004.

15. René Ranzinger and William S. York. 2012, Glyco-Bioinformatics today (August 2011) – Solutions and Problems. In Proceedings of the 2nd Beilstein Symposium on Glyco-Bioinformatics - Cracking the Sugar Code by Navigating the Glycospace. June 27th – July 1st, 2011, Potsdam, Germany. Pp. 107-130

16. About PostgreSQL, The PostgreSQL Global Development Group, "http://www.postgresql.org/". Retrieved on May 12, 2012.

17. Hibernate Overview, JBOSS Community. "http://www.hibernate.org/". Retrieved on May 11, 2012.

18. Java Beans, Wikipedia.Org, "http://en.wikipedia.org/wiki/JavaBeans". Retrieved on May 14, 2012.

19. HQL: The Hibernate Query Language, JBOSS Community, "http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html". Retrieved on May 12, 2012.

20. WebWork, Wikipedia.Org, "http://en.wikipedia.org/wiki/WebWork". Retrieved on May 02, 2012.

21. What is FreeMarker, FreeMarker Project, "http://freemarker.sourceforge.net/". Retrieved on May 02, 2012.

22. Introduction to Web Services, W3schools.com, "http://www.w3schools.com/webservices/ws_intro.asp", Retrieved on May 06, 2012.

23. Workflow, Wikipedia.Org, "http://en.wikipedia.org/wiki/Workflow". Retrieved on May 02, 2012.

24. Introduction to SOAP, w3schools.com. "http://www.w3schools.com/soap/default.asp". Retrieved on May 11, 2012.

25. Rest Web Services, Predic8, "http://predic8.com/rest-webservices.htm". Retrieved on April 29, 2012.

26. RESTful Web Services and constraints "http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm". Retrieved on April 28, 2012.

27. Introduction to XML, W3schools.com, "http://www.w3schools.com/xml/xml_whatis.asp". Retrieved on April 30, 2012.

28. Role of Web Services in SOA, Krawler Blog, "http://blog.krawler.com/2009/08/role-of-web-services-in-soa/". Retrieved on July 02, 2012.

29. WSDL 1.1 Vs WSDL 2.0, TheSoaTestingGeek, "http://thesoatestinggeek.wordpress.com/2012/03/12/wsdl-1-1-vs-wsdl-2-0/", Retrieved on June 02, 2012.

30. Welcome to Apache AXIS 2 /Java, The Apache Software Foundation, "http://axis.apache.org/axis2/java/core/". Retrieved on May 01, 2012

31. How Axis2 Handles SOAP Messages, The Apache Software Foundation "http://axis.apache.org/axis2/java/core/docs/userguide.html". Retrieved on May 01, 2012.

32. Web Service Description Language for Ja, Sourceforge.net, http://sourceforge.net/projects/wsdl4j/", Retrieved on May 04, 2012.

33. Overview JDOM API specifications, JDOM, "http://www.jdom.org/docs/apidocs/", Retrieved on May 15, 2012.

34. R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)," Web Semantics, vol. 6, 2008, pp. 61–69.

35. Semantic Annotations for WSDL and XML Schema, W3.org, "http://www.w3.org.com/TR/sawsdl/", Retrieved on May 02, 2012.

36. Style of WSDL, "http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/", Retrieved on May 04, 2012.

37. Invoking Web Services using Apache Axis2, Java.net, "http://today.java.net/pub/a/today/2006/12/13/invoking-web-services-using-apache-axis2.html". Retrieved on May 04, 2012.

38. HTTP Request methods for REST services. "http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods", Retrieved on May 15, 2012.

APPENDIX A

USER GUIDE

Registering a Web service in the Portal:

After specifying the generic details of a Web service in the first page of registration, we have the Web service definition page where we provide the WSDL (Web service Description Language), SAWSDL (Semantically Annotated WSDL) [35], WADL (Web Application Description Language) and SA-WADL (Semantically Annotated – WADL) definition files [34]. The definition files can be from a remote location, which is given as a URL, or a user's local file that can be uploaded.

The Web service definition page does not force the user to provide definition files. The files are provided only when it exists and if the user wants to upload. The WSDL files are taken for post processing to automatically populate the SOAP service generic information. The SA-WSDL, WADL and SA-WADL are just received and stored in the portal database.

Figure 25: Web service definition page

- Ports in Web service

After providing the WSDL file location, the WSDL file is parsed and the ports defined in the WSDL are retrieved and displayed to the user in the next page below. A service can have a number of ports defined under it. The user chooses the port to be registered with the portal. Only one port is associated with most of the Web services.

Figure 26: Web service Ports Page

- Prefilled generic details from WSDL

The WSDL file is parsed to retrieve the generic information about the service, which includes the Web service URL, and provider as shown below.

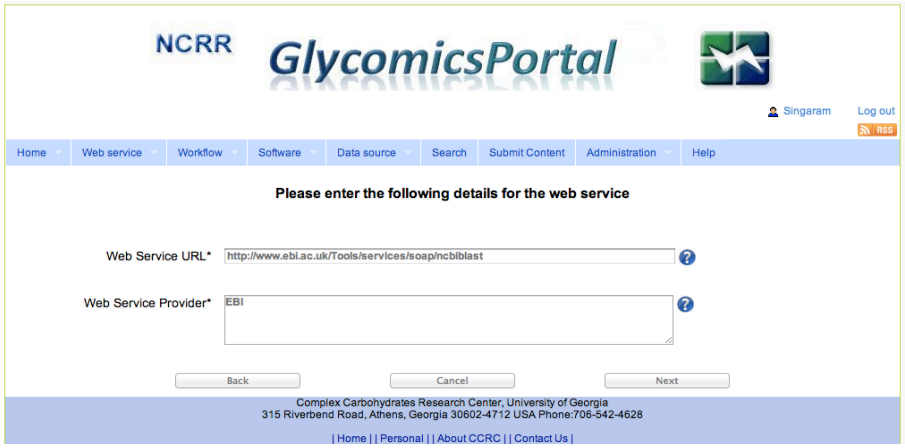

Figure 27: Web service generic details page

Verification page before finalizing and submitting the Web service is displayed to the user as shown below:

Figure 28: Confirmation page

Manage Entry page for the registered Web service

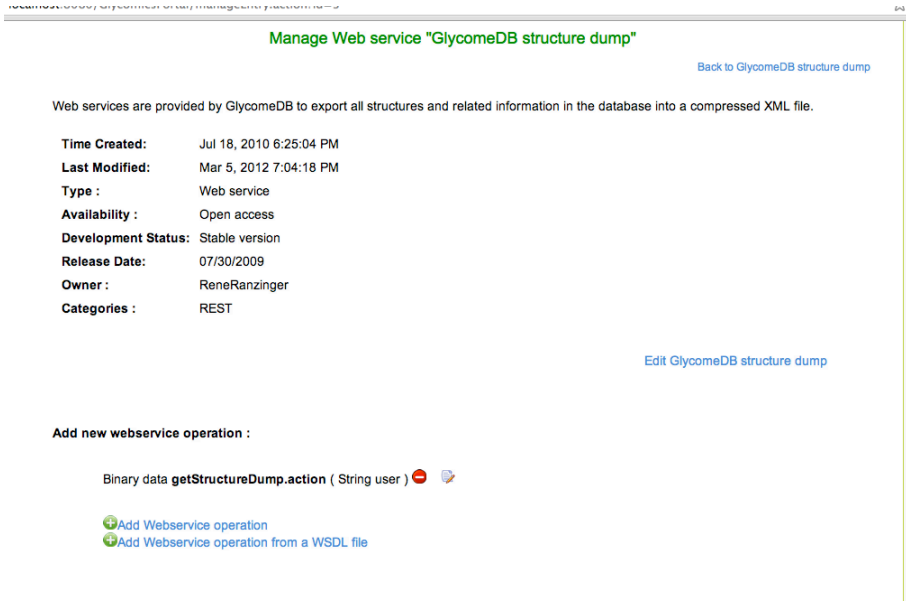

Figure 29: Manage Entry page

Automatic population of Web service operation based on the provided WSDL is initiated by clicking on "Add Web service operation from a WSDL file" as shown in figure 19.

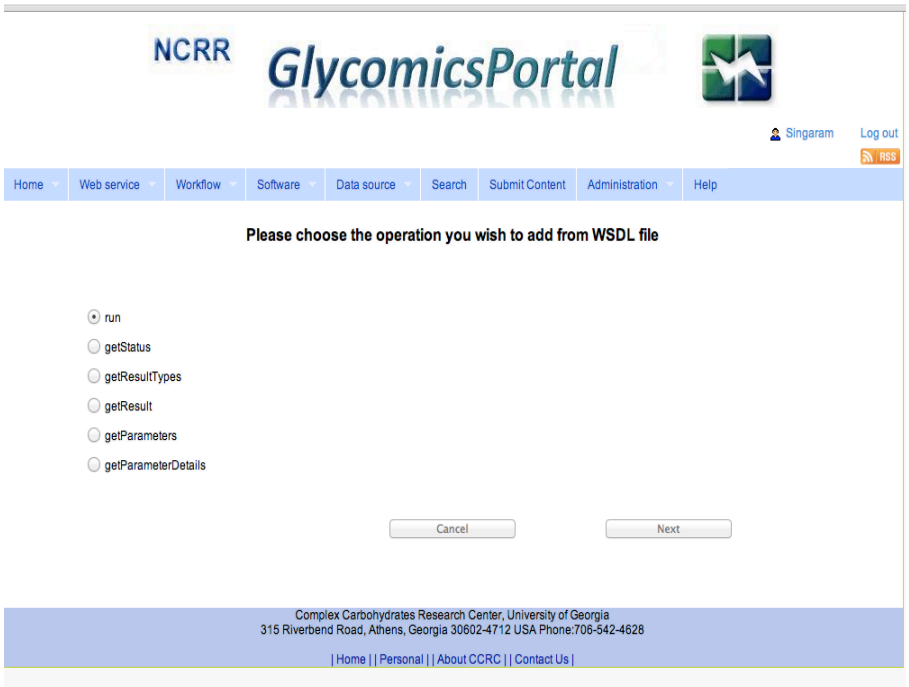

Figure 30: Automatic Population page

User chooses one of the operations listed in figure 30 from the WSDL. That prefills the generic details for the operation that include the Name, Description, Number of inputs, Number of outputs and Number of fault messages as shown below.

Figure 31: Prefilled generic details page

The user clicks on “Next” that takes the process to Input Parameters page with prefilled Name and Type of each parameter from WSDL. The user fills the Description manually.

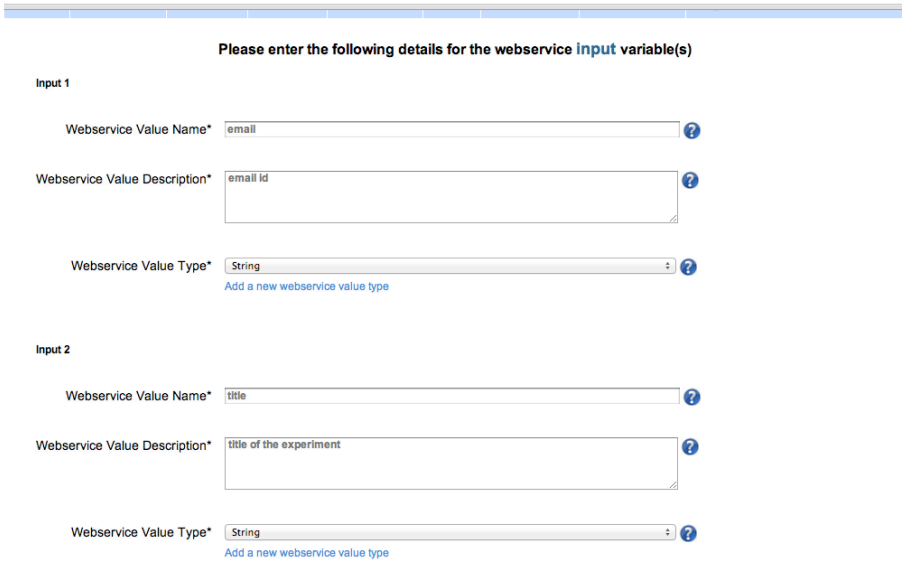

Figure 32: Prefilled input parameters page

Next page is the Output Parameters page that is prefilled with Name and Type of each output parameter from WSDL. The Description is entered manually by the user.

Figure 33: Prefilled output parameters page

Next page is the verification of entered information page where the user verifies the entered data and submits the operation to register it with the portal.

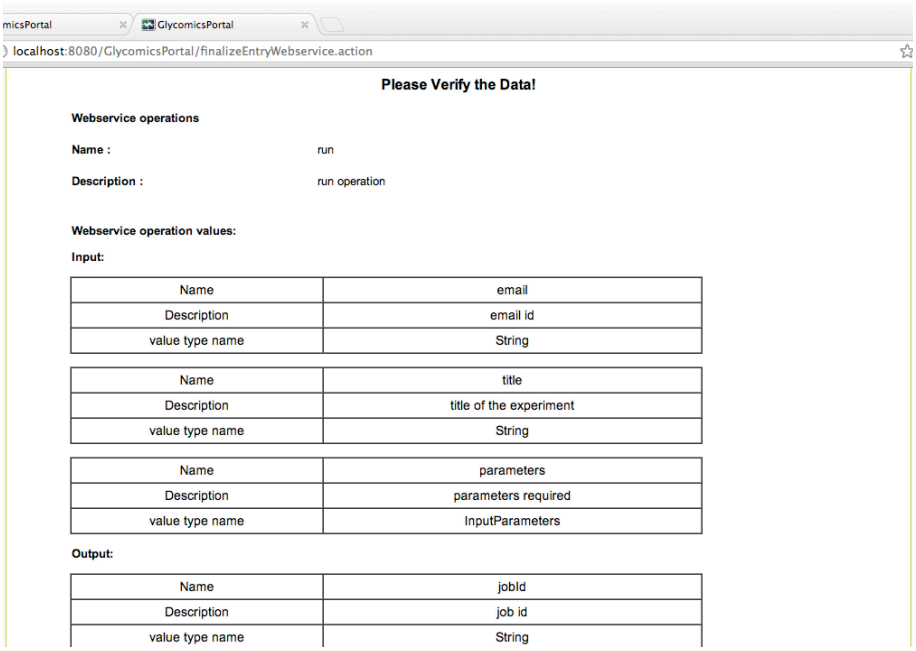

Figure 34: Web service Verification page

Refer to section 6.8 in chapter 6 for invocation of the registered web service in GlycomicsPortal.