LEVERAGING REST WEB SERVICES AND THEIR SEMANTIC EXTENSIONS FOR BIOINFORMATIC WORKFLOWS: A CASE STUDY USING GALAXY

by

SUMEDHA GANJOO

(Under the Direction of John A. Miller)

ABSTRACT

The field of bioinformatics involves analysis of large sets of data. This might entail leveraging of tools scattered over many Web sites. To provide the experimental biologists with a common platform capable of such analysis, this thesis focuses on extending a bioinformatics framework with Web service invocation support. Galaxy being substantially popular for its analysis tools and workflow management capability seemed like an ideal candidate to extend. This thesis proposes adding REST Web service support to Galaxy in a way that can be easily extended to SOAP Web services in the future. Also, it introduces an approach to add dynamic tools to Galaxy. To simplify the process of repetitive analyses on different sets of data, in this thesis we discuss enabling Web service invocation in the workflow portion of Galaxy. Also this thesis shows how we can leverage semantic annotations in Web services to improve the user's experience when interacting with Web services.

INDEX WORDS: Web services, REST, WADL, WSDL 2.0, SAWADL, Bioinformatics workflows, Galaxy, Semantic Web services

LEVERAGING REST WEB SERVICES AND THEIR SEMANTIC EXTENSIONS FOR BIOINFORMATIC WORKFLOWS: A CASE STUDY USING GALAXY

by

SUMEDHA GANJOO

B.Tech., College Of Engineering, Pune, Maharashtra, India

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

2010

LEVERAGING REST WEB SERVICES AND THEIR SEMANTIC EXTENSIONS FOR BIOINFORMATIC WORKFLOWS: A CASE STUDY USING GALAXY

by

SUMEDHA GANJOO

Major Professor:	John A. Miller
Committee:	Krzysztof J. Kochut Jessica Kissinger

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2010

DEDICATION

To Papa.

ACKNOWLEDGEMENTS

I would like to thank Dr. John Miller, for being an inspiring educator and an encouraging and supportive advisor over the past two years. I would also like to thank Dr. Jessica Kissinger for her time and valuable suggestions to improve this thesis. I thank Dr. Krzysztof Kochut for his time, and all the staff and faculty members of the department of computer science for directly and indirectly influencing this work.

This work would have not been possible without the pep talks and the suggestions that I needed from time to time. And for all that and more, I thank my family and friends. Last but certainly not the least; I thank God for His blessings and guidance.

TABLE OF CONTENTS

Page

CHAPTER 1

INTRODUCTION

1.1. Introduction to Galaxy

Galaxy [1, 2] is an Internet based framework which provides an analysis platform for performing analyses on different types of bioinformatics data obtained from a variety of bioinformatics databases and resources. Apart from providing various analysis tools it also provides a workflow component which lets the user save workflows and share them with colleagues. It is highly popular in the bioinformatics community due to its simple and easy to use interface and the fact that it allows users to connect to multiple resources.

Scientists can register at, and use, the globally hosted instance of Galaxy at http://main.g2.bx.psu.edu/, or download and setup a local instance of Galaxy server from http://bitbucket.org/galaxy/galaxy-central/src. Galaxy is an open-source project funded by NIH, NSF, Penn State, Emory, and the Pennsylvania Department of Public Health. Further development of the project can be followed at its wiki: http://bitbucket.org/galaxy/galaxy-central/wiki/Home.

1.2. Motivation

There are a myriad of Web services available on the Web to aid biologists in processing and acquiring data. Major service providers for these Web services are EMBL - European Bioinformatics Institute [3], DNA Data Bank of Japan (DDBJ) [4], Kyoto Encyclopedia of

Genes and Genomes (KEGG) [5] and National Center for Biotechnology Information (NCBI) [6]. As noted on July 1st 2010, BioCatalogue [7] listed 120 such service providers and 1,695 services (1,638 SOAP Web Services and 57 REST Web Services) hosted by them. EMBL-EBI itself hosts 11 REST only, 11 SOAP only and 24 REST/SOAP bioinformatics related services.

Currently, Galaxy has no support for using these Web services. We focus on adding REpresentational State Transfer (REST) [8] Web service capability to Galaxy, so that the users can easily use the many readymade REST Web services for bioinformatics analysis available on the Internet. Adding Web service functionality to Galaxy is only a logical extension of Galaxy's aim to provide a common framework to biologists [9] and save them the effort required to move the data around various resources/tools. Currently, mostly all the tools usable through Galaxy are hosted on the Galaxy server. Galaxy also allows the user to access data from some other specific Web sites registered with Galaxy. Another way of importing outside data to Galaxy is by uploading an external file saved locally.

The extensions to Galaxy made in this thesis aim at allowing inclusion of Web services in Galaxy workflows like other Galaxy tools. This implies that data output from a Web service could be fed to a Web service or a tool in Galaxy and similarly, a tool's output could be fed to a Web service as input, making the interaction far simpler and automated for the user. This saves the user the need to go to various Web sites individually to perform required analyses. To enable Web services to be incorporated in workflows in Galaxy, it is required to provide Galaxy with the flexibility of interacting with any Web service provider in two ways:

i. Accessing Data: Allowing users to access sources beyond those that are currently registered with a Galaxy server via Galaxy's interface.

ii. Performing Analysis: Allowing users to invoke/use tools remotely rather than requiring them to be installed and registered on a Galaxy server.

Thus, we work on adding extensions to Galaxy to enable its interaction with various REST Web services, to enable a bioinformatics framework that can leverage REST Web services available on the Web.

1.3. Outline

Chapter 2 talks about REST Web services in general followed by an introduction to technologies essential for understanding the project. It also acknowledges previous work done in the field of extending Galaxy with Web services. In chapter 3, we give an overview of the architecture of the Galaxy code base, and discuss various approaches to extend Galaxy. Chapter 4 discusses the capabilities added to Galaxy as a part of this project, followed by chapter 5 in which an introduction to generic and extended workflows in Galaxy is given. In chapter 6, the system is evaluated with the help of an example workflow. Chapter 7 gives a summary of the work and proposes future extensions to it. Appendices A, B and C are the installation, user's and developer's guides, respectively.

CHAPTER 2

BACKGROUND

2.1. Introduction to REST Web Services

Web services [10] enable interoperation between different software applications, irrespective of frameworks and platforms. According to W3C, a Web service is "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web service in a manner prescribed by its description" [10]. This description can be realized by the use of an XML document following one of the following specifications: Web Application Description Language (WADL) [11, 12] or Web Services Description Language (WSDL) [13, 14].

WADL is designed to provide a machine processable protocol description format for use with HTTP-based Web applications, especially those using XML. Mostly, WADL documents are used to describe REST [8] Web services and WSDL documents to describe SOAP [15] Web services. Although WSDL 1.1 [13] does not support the description of REST Web services, WSDL 2.0 [14] is designed with support for REST Web service description. Our research focuses on REST Web services described using WADL as well as WSDL 2.0.

REST is a resource-centric Web service design model, with each resource being represented by a URI (Uniform Resource Identifier) [16]. The basic principles followed by the REST approach are the following:

i. Use of the HTTP Web Protocol: To create, retrieve, update or delete a resource a REST Web service always uses one of the HTTP methods: POST, GET, PUT or DELETE. A REST Web service does not have any other operations apart from these four.

ii. Statelessness: REST architecture is said to use "stateless" operations because a request to an operation contains all the information needed to generate a response independent of the session.

iii. Use of URI: A REST Web service identifies each resource by a URI. To make the URIs more intuitive, a hierarchical structure is followed while defining them. This also makes it easier to travel from one resource to other using hyperlinks.

iv. XML message format: REST Web services typically work with XML data formats for messages to pass request/response payloads.

2.1.1. Web Application Description Language (WADL)

According to W3C WADL is "designed to provide a machine processable description of HTTP-based Web applications" [11]. A WADL document can be used to describe all the accessible resources of the resource-centric Web service design model of REST. The only methods allowed in a WADL document are the HTTP methods GET, PUT, POST or DELETE. All WADL elements have the XML namespace name: http://wadl.dev.java.net/2009/02. Next is a brief overview of the basic tags followed by a sample WADL document.

The *application* element forms the root. It contains zero or more *resources*, each of which contains zero or more *resource* elements. Each *resource* element can again contain zero or more *resource* elements representing the sub-resources. A URI for a resource is generated by

combining the base URI obtained from the *resources* tag and appending it with the *path* attribute of the *resource* tags in hierarchical order. Once a resource is defined, the *method* tag is used as a child element to define the HTTP protocol methods applicable to that resource. Each *method* has a *request* and a *response* tag describing the input and the output to the method. The request to a method contains a set of parameters described using the *param* tag. A parameter can be specified as a required parameter by setting the attribute *required* for the *param* to true. Also a *default* attribute of *param* exists that allows specification of a default value for the parameter. Figure 1 shows an example of a WADL document obtained from EuPathDB [17].

```
<application xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd">
  <resources base="http://eupathdb.org/webservices/">
    <resource path="GeneQuestions">
      <resource path="GenesByTextSearch.xml"></resource>
      <resource path="GenesByTextSearch.json"></resource>
      <method name="POST" id="genesbytextsearch">
        <doc title="display_name">Text (product name, notes, etc.)</doc>
        <doc title="summary"></doc>
        <doc title="description"></doc>
        <request>
          <param name="text_search_organism" type="xsd:string" required="true"></param>
          <param name="text_expression" type="xsd:string" required="true"></param>
          <param name="text_fields" type="xsd:string" required="true"></param>
          <param name="whole_words" type="xsd:string" required="true"></param>
          <param name="max_pvalue" type="xsd:string" required="true"></param>
          <param name="timestamp" type="xsd:string" required="false"></param>
          <param name="o-fields" type="xsd:string" required="false" default="none"></param>
          <param name="o-tables" type="xsd:string" required="false" default="none"></param>
        </request>
        <response>
          <representation mediaType="text/xml"/>
          <representation mediaType="text/json"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

Figure 1: Genes by text search WADL from EuPathDB [18]

2.1.2. Web Service Description Language Version 2.0 (WSDL 2.0)

According to W3C, Web Services Description Language "provides a model and an XML format for describing Web services". It separates the functionality and the messages from the concrete details like message format and network protocol [13, 14]. WSDL 2.0 [14], unlike WSDL 1.1 [13], supports both REST and SOAP Web services.

A WSDL [13, 14] document has several elements, which together are used to specify the service's URL, communication mechanisms, operations offered and the structure of the input and output messages. One of the elements of a WSDL document is the binding element, which is used to specify the communication mechanism supported by the Web service. As seen in the beginning of section 2.1, REST services communicate via HTTP, so a service capable of REST style behavior is declared by specifying the binding type to be HTTP in WSDL 2.0. This can be done by using the namespace "http://www.w3.org/ns/wsdl/http" as the value of the type attribute of the binding element. Any service with an endpoint with a HTTP binding is capable of REST style behavior.

2.1.3. Semantically Annotated Web Application Description Language (SAWADL)

Battle and Benson in their work on semantic extensions to REST [38], proposed ideas to semantically annotate WADL and introduced SAWADL [38]. One of the features of SAWADL is that every input parameter can be annotated with a concept from an ontology. This is done by using the *modelReference* attribute of the *param* tag.

The basic additions to a WADL for annotating an input parameter with a concept would be to the application and the param tag. The application tag in the WADL is changed to specify the ontology reference and the SAWADL specification location in it, e.g., add the following

attributes with appropriate values: xmlns:Ontology1=http://www.owl-ontologies.com/ontologyName.owl and xmlns:sawadl=http://www.standards/sawadl/spec/sawadl# to the application tag. Later in the param tag a reference to the concept can be specified by using the modelReference attribute, i.e., add sawadl:modelReference="Ontology1# ConceptName" to the param tag. The values for the attributes above are merely examples and should be changed appropriately.

2.2. Related Work

This project apart from providing new approaches, also extends the approach introduced in Shefali Shastri's work [19] for extending Galaxy. She added a REST Web services extension to Galaxy by proposing one generic tool capable of invoking all REST Web services. The drawback of this tool was that the user was assumed to have all the needed information for invocation of the Web service like parameter names and values.

2.3. Technologies Leveraged

This section gives a brief introduction to the existing software used by the project. Firstly, the various parsers used to read the documents describing REST Web services are talked about. Next is an introduction to JPype [20], followed by a little background knowledge of the OWL API [21].

2.3.1. Description Document Parsers

We extend Galaxy to provide support for REST Web services described using a WADL, WSDL 2.0 or SAWADL document*. Thus, to be able to parse these documents we need parsers

*In future, to support WSDL 1.1 and SAWSDL 1.1 and SAWSDL 2.0 standards we can use WSDL4J, SAWSDL4J or Woden4SAWSDL4J parsers, respectively.

for documents built on all three specifications. The WADL parser [22] used is obtained from the Large Scale Distributed Information Systems (LSDIS) lab, at UGA. This parser takes the location of a WADL document and reads various components of the document. The SAWADL parser [22] is also a contribution made by the LSDIS lab, at UGA. The major difference being that the SAWADL parser also reads the annotations, if any, in the description document.

To parse WSDL 2.0 documents the Apache Woden API [23] is used. The Woden API consists of Java interfaces to read, write and edit WSDL 2.0 documents. To further study Woden refer to the user guide available at http://ws.apache.org/woden/userguide.html.

2.3.2. JPYPE

The parsers described above are all implemented in Java. Galaxy tools added by us being implemented in Python cannot directly invoke these parsers. JPype [20] acts as a middle layer of interaction between Python and Java. The JPype project is built with an aim of allowing programs written in Python complete access to Java libraries, by interfacing the Java Virtual Machine (JVM) and the CPython Virtual Machine. It allows setting up of a JVM from a Python program, with a specified class path and jar path. For a better understanding of the JPype project please refer to http://jpype.sourceforge.net/index.html.

Some of the other projects enabling Python Java interaction are Jython [24], JEPP [25] and JCC [26]. Jython re-implements Python in Java. However, it does not allow access to some modules in Python's standard library. JEPP stands for Java Embedded Python. It embeds CPython in Java. JEPP provides a scripting solution for Java, enabling Java code to run existing Python scripts. It does not allow Python code to invoke a JVM though. JCC enables calling Java

code from C++/ Python via Java's Native Invocation Interface (JNI). Because of JPype's easy to learn nature it is favored over the other available options.

2.3.3. OWL API

An Ontology can be defined as a representation of some defined terms and their relationships with one another. Using these ontologies, documents can be made to have clear semantics and hence be machine interpretable. OWL (Web Ontology Language) [27] is a language for describing the terms (also called concepts) and relationships in an ontology.

The OWL API [21] is an open source project providing a Java API for creating and manipulating OWL Ontologies. To be able to exploit the features of an annotated REST Web service described using a SAWADL document, we use the OWL API to read and parse the ontology and access required information from it. From a SAWADL document the concepts that the input parameters are annotated with can be obtained from the *model reference* attribute of each parameter. The OWL API can be used to obtain specific information like the description or comments attached to that concept from the ontology. The Javadoc for OWL API can be found at http://owlapi.sourceforge.net/javadoc/index.html and for other documentation please refer http://owlapi.sourceforge.net/documentation.html.

CHAPTER 3

APPROACH

Before describing various approaches to extend Galaxy to support Web service invocation, it is important to understand the architecture of Galaxy, its key features and restrictions. This chapter gives a brief overview of the architecture of Galaxy followed by the various ways of making changes to Galaxy to incorporate support for executing Web services through Galaxy.

3.1. Galaxy Architecture

The two major sections of our concern from the Galaxy code are named “lib” and “tools”. The tools section is a placeholder for various existing tools provided in Galaxy. The lib section consists of all the server side Python code for the implementation of the Galaxy framework. Thus, to edit or add a feature that is a part of the behavior of Galaxy’s framework: its internal working and implementation, modules in the lib section would be altered. Whereas any behavior restricted to a particular tool in Galaxy would require additions or modifications to the tools section. Figure 2 gives a brief overview of these two sections, focusing on modules that need to be changed in Galaxy for our Web service extensions.

All Galaxy tools are stored under “tools” under Galaxy’s main directory [34]. Each tool is conventionally added as a directory containing some XML and Python files. The XML file describes the tool’s interface and also specifies the Python module that is invoked by the tool. Apart from tools the other directory that we focus on in Figure 2 is lib. It represents the library

containing most of the Python code base required for setting up, running and maintaining of Galaxy server.

For Galaxy to recognize any tool, the tool_conf.xml file needs to be updated with the location of the tool [34]. This file is present directly under the main Galaxy directory.

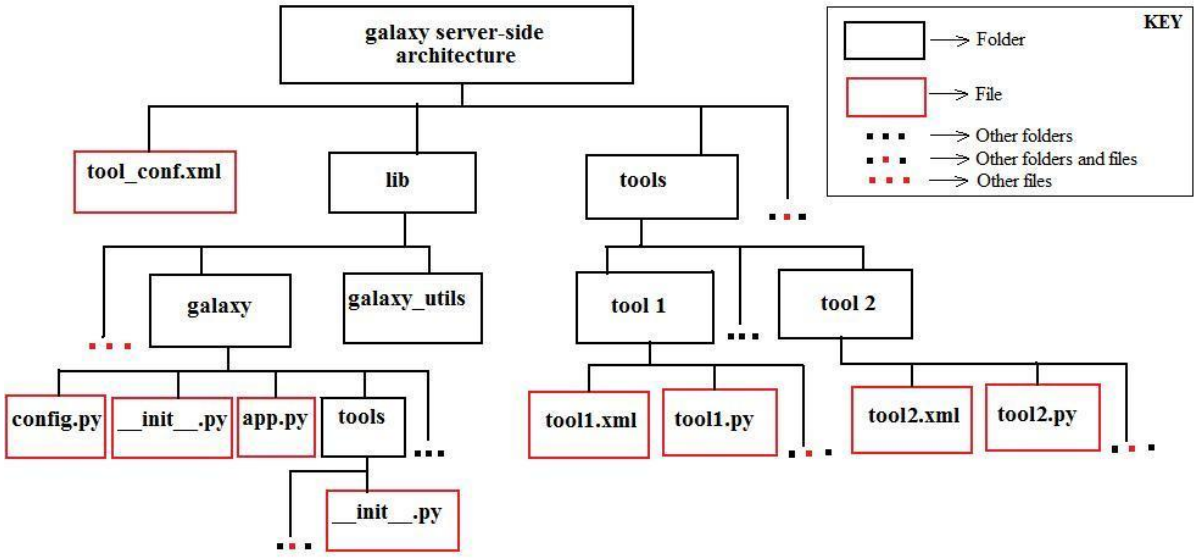

Figure 2: Galaxy's architecture

Based on the amount of modifications made to Galaxy's implementation, we define three approaches to extend Galaxy to provide Web service support. These approaches are also distinct in the nature of changes and extensions they allow to be made to Galaxy. These approaches showcase various levels of freedom varying from allowing modifying the main Galaxy source code itself to extending Galaxy only by adding tools to the framework.

Galaxy, being an open source project, the former approach of modifying the source code itself seems more natural. It also provides the most flexibility and scope for extension. However, these extensions are much more difficult to maintain across new releases of Galaxy. The latter approach allows extensions only by adding tools, in our case a universal Web service client, to

Galaxy. The downside of this approach is that the client's interface is restricted by the definition of a tool's behavior in Galaxy. The upside is that no maintenance issues arise in this approach.

We propose an intermediate third approach that apart from adding tools allows minimal changes to Galaxy's source code. This final approach seems best suited for the purpose of providing Web service support to Galaxy.

We develop REST Web service extensions into Galaxy following both the "Restricted Tool Addition" approach as well as the "Minimal Change" approach. The three approaches for extending Galaxy are discussed next.

3.2. Maximal Functionality Approach

The Maximal Functionality approach implies making changes to the code base of Galaxy to incorporate the behavior of handling Web services through the Galaxy interface. The extensions to Galaxy to support semantic Web services, by Rui Wang [28], demonstrate the ability to add Web services directly to the core of Galaxy.

This approach requires extensive development. Apart from the tools that need to be added, it adds around two thousand lines of code scattered over nine existing packages and two newly added packages. In order to provide more capabilities and an improved user interface significant changes to the Galaxy code base are required; some of these changes are described below [28]:

1. Defining a new type of tool for handling Web services.
2. Adding the ability to dynamically add tools of this newly added type. Galaxy has a module called "runner" that dictates the behavior of the tool at run time. By default

Galaxy tools are all static in nature. A new "runner" is defined and invoked for tools handling Web services to add dynamic behavior.

3. Changing the interface (module tool_menu.mako) so that it refreshes automatically and displays the newly added tools.
4. Adding the ability to invoke Web services from Galaxy tools. Depending on the Web service the inputs to this tool vary and the tool has to act as a specific Web service client.

Though the above approach provides more flexibility and an improved user experience, the main drawback is that it is difficult to maintain across different versions of Galaxy. Unless the Galaxy developer team adopts the changes and incorporates them in the main instance of Galaxy, installing these extensions into a newer Galaxy version is quite challenging. Even if support for these extensions is provided in the main instance of Galaxy, inclusion of new features will require changing the code base again.

3.3. Restricted Tool Addition Approach

The Restricted Tool Addition approach allows extending Galaxy only by adding tools. No changes are made to the main Python code base. This approach is the easiest to implement as it requires knowledge only of writing tools in Galaxy and does not require the developer to understand in detail the implementation of the framework. In this approach, we extend the Universal REST client [19] added to Galaxy as a tool. The extended Galaxy architecture is shown in Figure 3 below.

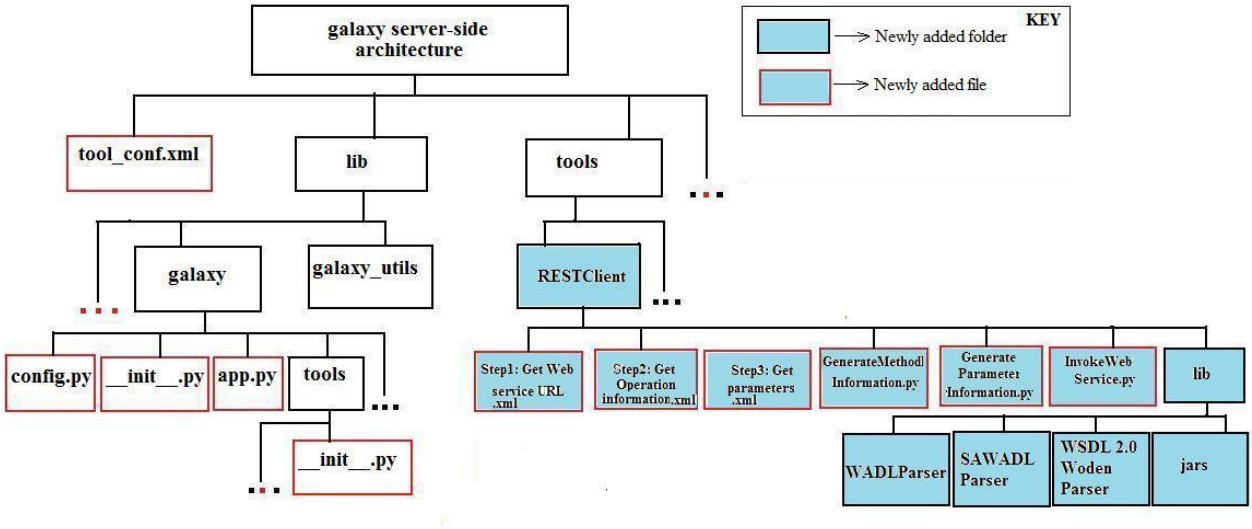

Figure 3: Architectural changes by Restricted Tool Addition approach.

The REST client tool shown in Figure 3 represents a Universal REST Client. It is called so because given adequate information about any REST Web service like its description document it acts as a client capable of invoking that Web service. A major advantage of this approach is that it requires minimum maintenance effort across versions of Galaxy. Galaxy is built such that tools can be plugged into any version of Galaxy easily, by just modifying the tool_conf.xml and adding the tool specific code and XML files in the tool directory.

The main drawback of this approach is that the behavior of the tool added is restricted by Galaxy's definition of a tool. As Galaxy did not support interaction with Web services originally, the Universal REST client added as a tool is limited in nature. It is harder to learn to use and does not have a very user-friendly interface.

3.4. Minimal Change Approach

The Minimal Change approach allows making minimal changes to the Galaxy code base apart from allowing addition of tools to Galaxy. It differs from the Maximal Functionality

approach in the extent of changes it allows to be made to the Python code base. Unlike the Maximal Functionality approach, the only feature support that is added to the core code-base directly is the ability to reload the toolbox without having to restart the server. As can be seen in Figure 4, this requires only changing two Python modules (Appendix C), i.e., lib/galaxy/app.py and lib/galaxy/tools/__init__.py from the core Python code base of Galaxy. These changes enable adding tools to Galaxy dynamically by adding the capability to reload the toolbox in Galaxy without having to restart the server.

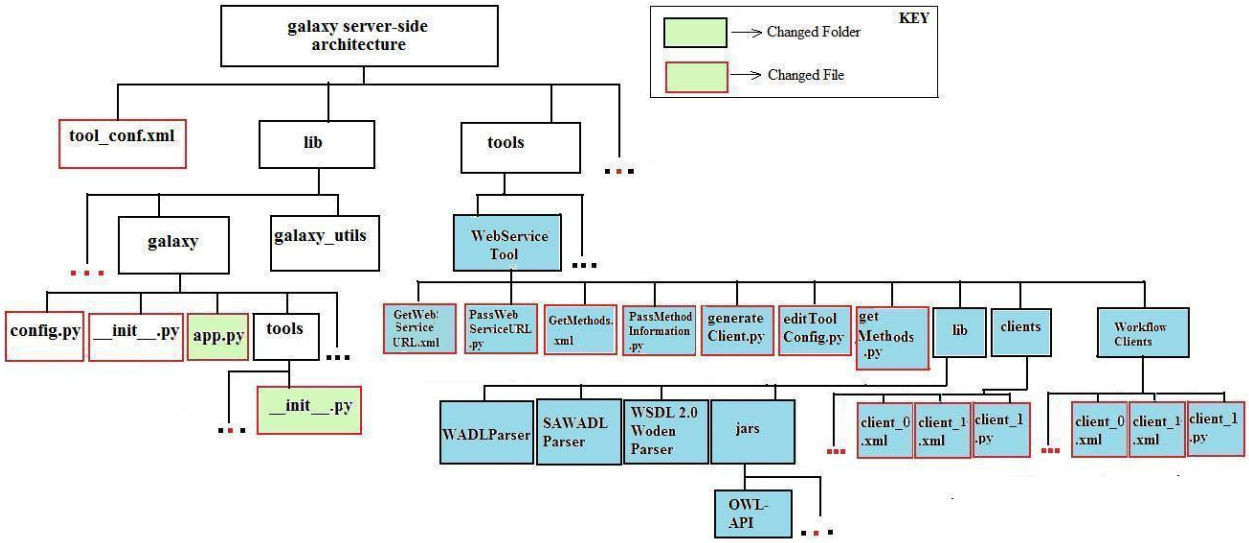

Figure 4: Architectural changes by Minimal Change approach.

Apart from the changes made to the code base, this approach needs the addition of a tool for registering Web services with Galaxy. For every registered Web service a Web service client is added as a tool to Galaxy. This Web service client can then be used to invoke the Web service with specific parameter values. A flow diagram of the design of the Minimal Change approach is shown in Figure 5. Unlike the Restricted Tool Addition approach where a universal client was

added, here we add Web service-specific clients as tools to Galaxy. Thus, following this approach allows for a better user experience as discussed and shown in chapter 6.

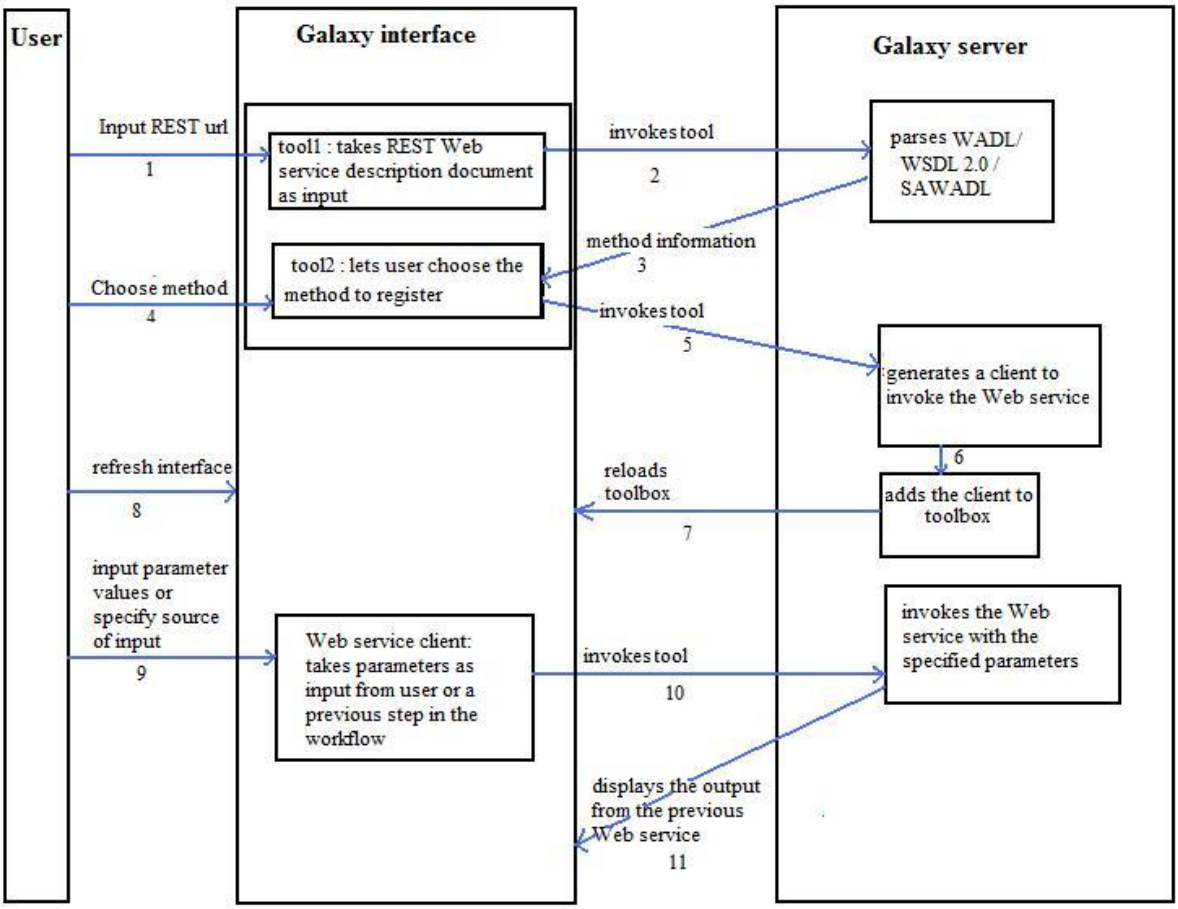

Figure 5: Design of Web service extensions using Minimal Change approach.

Another advantage of this approach is it allows for extending the Web service support in the future without having to make any changes to the main code base. Also it is easy to maintain across newer Galaxy versions as the modifications to the source code are minor and easy to integrate.

Our Web service addition tool does not add generic tools to Galaxy, but only tools that act as Web service clients. The code that runs on executing these added tools invokes the Web service which is on a third party system and hence cannot harm the Galaxy server hosting the tool. This ensures there is no breaching of security. Also the working of the tool can further be easily modified to add access control, allowing only users with certain privileges to add tools dynamically.

CHAPTER 4

CAPABILITIES ADDED

This chapter discusses the various features added for the Web service extensions made to Galaxy. As discussed before, the main purpose of the project is to enable scientists to easily access Web services via Galaxy.

Below we discuss the capabilities provided by our Universal REST client as well as the Web service-specific clients. The Universal REST client is added using the Restricted Tool Addition approach, whereas the Web service-specific clients are added using the Minimal Change approach. As the Minimal Change approach is more flexible some of the features discussed below are limited to the Web service-specific clients, whereas some like parameter entry and semantic support, are inbuilt in both the Universal as well as the Web service-specific clients.

4.1. Parameter Entry

To make the process of Web service invocation through Galaxy easier for the users, the Web service clients parse the description document for a REST Web service, i.e., a WSDL 2.0, WADL or SAWADL document to obtain the input parameter information. The user sees this information and specifies parameter values for selectable options.

In the universal client approach the user can choose to display additional information corresponding to each parameter's name. This additional information consists of the parameter's

data type, default value and if it is required or not. The only problem being here that as the Universal REST client is added using the Restricted Tool Addition approach it is a static tool, and requires the user to iterate through the parameter entry process to enter each parameter value.

Whereas, Web service clients added as tools using the Minimal Change approach are dynamic in nature. To enable ease of parameter entry without looping, we built the clients such that each Web service input parameter is read from the specified WADL/ WSDL 2.0/ SAWADL and displayed as a tool input parameter. A sophisticated and easily comprehensible interface is provided. Instead of having to display additional information about each parameter as an option on the tool form, additional information, like the type, is displayed as a tip in the help section. This works in accordance with Galaxy's style of using the help section in their tools and makes the clients easier to use. Figure 6 shows how parameter information is used to simplify the use of clients.

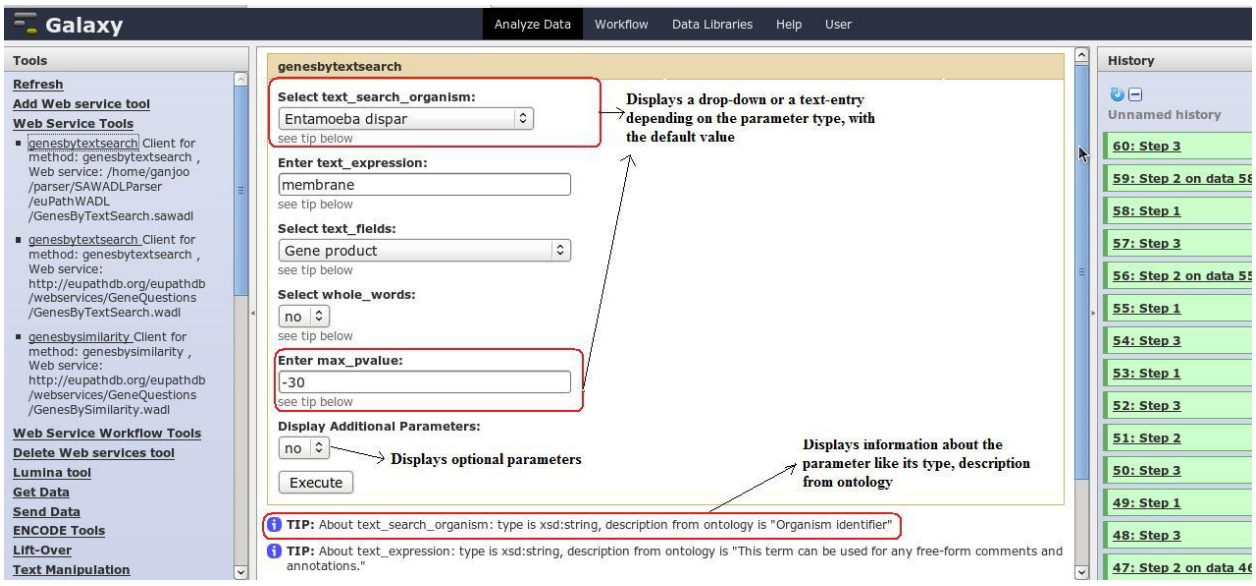

Figure 6: Leveraging parameter information from SAWADL in a Web service-specific client

4.2. Universal Client

The Universal REST client [19] has been added as a tool to Galaxy. It is capable of invoking any REST Web service given a URL locating a document describing the REST Web service. This description document could be following any of the following specifications: WADL, WSDL 2.0 or SAWADL. As it is built using the Restricted Tool Addition approach the Universal client is added as an independent tool to Galaxy, and does not require any additions to Galaxy's source code. Figure 7 gives an overview of the interface of all the steps of the REST client.

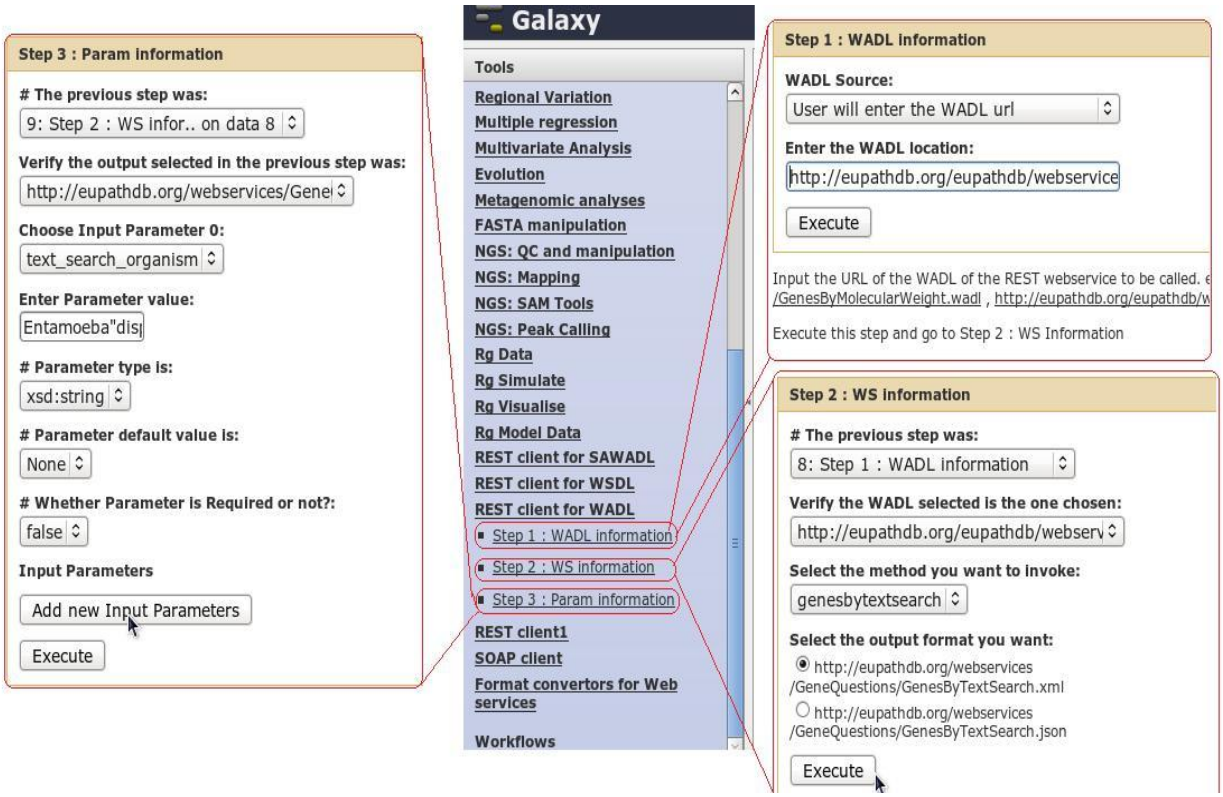

Figure 7: Three steps for using the Universal REST client

4.3. Web Service-Specific Client

Using the Minimal Change approach we add the Web Service Addition tool to Galaxy, capable of dynamically registering Web service-specific clients as individual tools. This requires

parsing the given description document, creating a tool specific for invocation of the chosen Web service operation and refreshing Galaxy's toolbox without having to restart the server. To enable dynamic addition of tools via the Galaxy GUI, we not only added a new Web Service Addition tool, but also modified the Galaxy's source code to add the toolbox refresh capability to Galaxy.

4.3.1. Refresh Capability

To enable adding tools dynamically to Galaxy, we incorporated in Galaxy the ability to refresh and reload the toolbox without having to restart the server. This required additions to the Galaxy source code to add the reloading of toolbox feature. This feature is provided as a part of the Web service registration tool. Currently, it is not a password protected functionality and anyone who has access to the tool can refresh and reload the toolbox through the GUI.

4.3.2. Web Service Addition Tool

To enable addition of REST clients dynamically via the Galaxy GUI, a Web service addition tool is provided. It lets the user register a Web service client as a tool through the GUI. Like the Universal REST client it works with REST Web services described using any of the WADL, WSDL 2.0 or SAWADL documents. It asks for the user to choose one of the operations specified in the description document and generates a client, tailored to that Web service operation, as a tool in Galaxy. An overview of the interface of the tool is shown in Figure 8. A detailed guide on the use of this tool can be found in Appendix B.

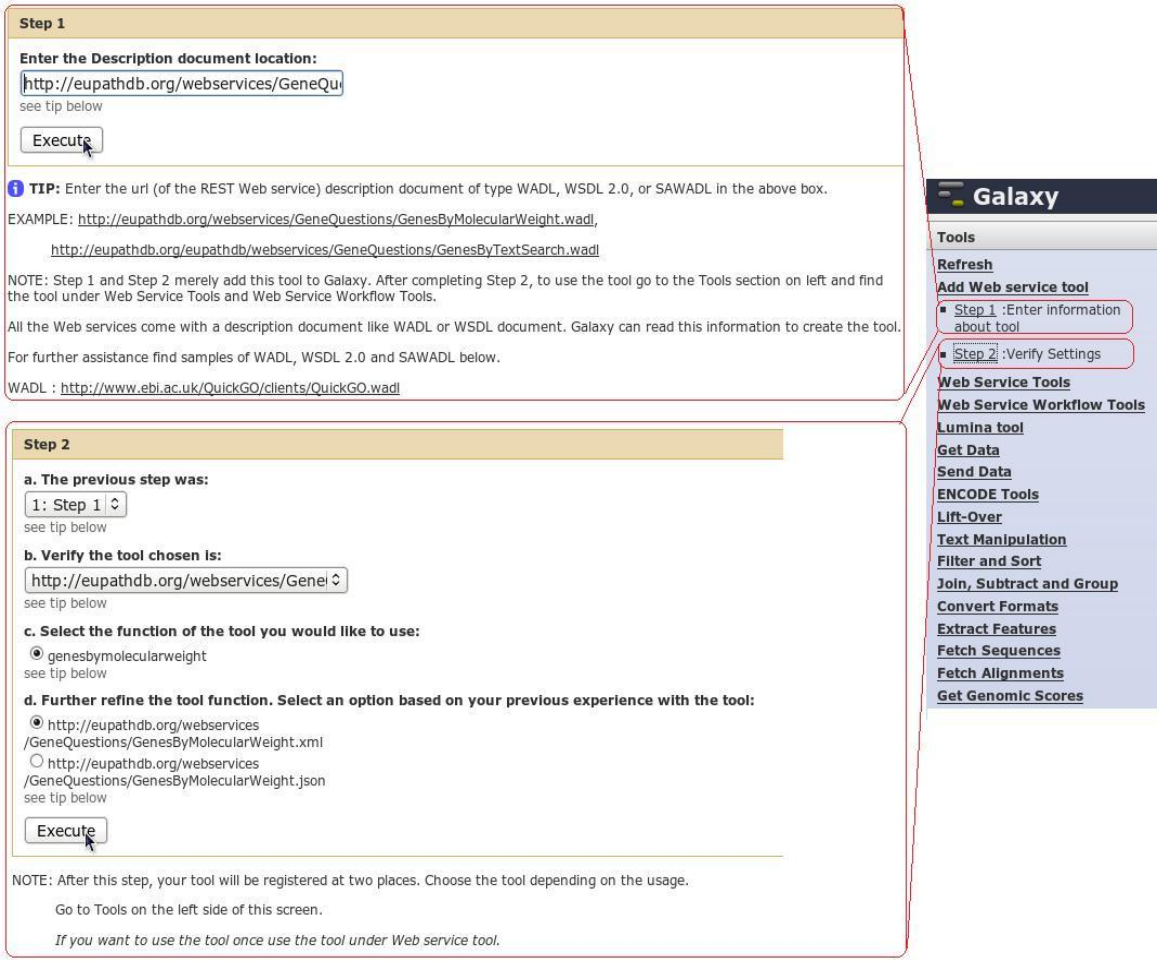

Figure 8: Add Web service tool overview

4.3.3. Dynamically Generated Web Service-specific Clients

Though the Universal REST client lets the user invoke any REST Web service provided its description document, it is not tailored to be specifically used for that Web service and hence does not have an easy to use interface. Using the Minimal Change approach we can add Web service-specific REST clients to Galaxy as individual tools. These tools are much easier to understand and use. These clients can be added as tools in Galaxy dynamically through the GUI. These tools take as input the values of the mandatory parameters for the invocation of the Web service. Optionally, they can also take the rest of the parameters as input.

4.4. Workflow Support

Galaxy is popular for its easy-to-use workflow management system. The tools in Galaxy can be incorporated in workflows which can be run on large sets of data, avoiding repetitive analytical steps to be performed by the user. Also, Galaxy allows the users to share workflows and histories with one another. Thus, it is important to make the Web service clients pluggable in Galaxy workflows. To enable this we add special clients as tools to Galaxy which let the user choose output from a previous step in the workflow as an input to a parameter. An in-detail discussion of Web service support in Galaxy workflows can be found in the Chapter 5.

4.5. Semantic Extensions

To help the scientists understand the meaning of the Web service parameters we provide support for REST services described using SAWADL. SAWADL allows for the input parameters to be annotated with concepts from an ontology. If a SAWADL document is used for describing the REST Web service, with each input parameter apart from its type and default value the client also displays the comment, if any specified in the ontology, about the concept annotating the parameter.

Also semantics are being leveraged for Web service discovery. A Web service discovery tool integrated in Galaxy will help users find the Web services which they can then register into Galaxy. Web service discovery support in Galaxy is discussed more in detail in section 7.2.1.

CHAPTER 5

WORKFLOWS

A workflow can be thought of as a series of computational steps executed one after another. The field of bioinformatics is a major ground for the application of workflows, as here step-by-step analysis of huge amounts of data is required, repetitively. Currently there exist a number of Workflow Management Systems to help the biologists in creating and executing workflows. Galaxy is one such Workflow Management System that works with tools installed on Galaxy's server. Some other Workflow Management Systems are BioExtract [29, 30], Taverna [31], Kepler [32]. This chapter gives a brief overview of the workflows in Galaxy followed by ways to incorporate Web services in them.

5.1. Workflows in Galaxy

Galaxy supports the creation and execution of workflows. It provides numerous tools for analysis of large sets of data, for example tools to manipulate multiple alignments, compare genomic annotations, compute length of and filter FASTA sequences [33]. As each step of analysis is recorded in Galaxy's history, scientists can reproduce an experiment as a workflow from a saved history. Also Galaxy allows scientists to share their histories.

Two ways of generating workflows in Galaxy are either from scratch or from the history. To create a workflow from scratch, the user can pick and drop tools from Galaxy's toolbox on to the Galaxy's workflow canvas. These tools can then be connected by the user directing the

output of one into the input of another. A second way of creating workflows is from history. The user can generate workflows from history by merely a click, and save the desired analyses as a separate workflow.

5.2. Workflow Support in Web Service Tools

It is essential to make the Web service clients pluggable in Galaxy workflows. A user can then form a workflow with multiple Web services or with a combination of Web services and Galaxy tools.

Each Galaxy tool asks for some input values from the user before executing. The nature of these inputs determines the tool's capability to take part in workflows. If the inputs are directly to be entered by the user or chosen from a set of constant values, then on the workflow canvas the tool would not have an input slot and the output of no other tool would be able to be fed to the input of this tool. If on the other hand, the tool is written in a way that allows the user to choose the output of a previous step as an input while executing, the tool would show an input slot on the workflow canvas and another tool's output could be fed to this tool's input (see Appendix C).

To make the Web service clients easily pluggable in the Galaxy workflows they should be built to accept outputs from previous steps as input parameters, and not just user entered values. Thus, in the Minimal Change approach we add two types of tools to Galaxy. One that can take input from a previous step and are workflow compatible; and a second for independent invocation of Web services through Galaxy or for use as a first step in a workflow. The use of the workflow compatible clients in Galaxy is further explained with the help of an example in chapter 6.

CHAPTER 6

EVALUATION AND EXAMPLE WORKFLOW

To evaluate the extensions made to Galaxy we tested the Universal REST clients as well as the Web service-specific clients with various REST Web services provided by EBI [3] and EuPathDB [17]. Both EBI and EuPathDB host a wide ranging collection of Web services. For testing with EuPathDB services we used the WADL documents available on their Web site, but for EBI services we created the WADL documents from the documentation provided on their Web site [17].

With the help of an example workflow, we show the significance and use of adding the REST Web service extensions to Galaxy. Galaxy being a highly popular bioinformatics analysis framework seemed like an ideal candidate for extension. Not only does it have an easily learnable interface, many biologists are already familiar with the working of Galaxy's tools and workflow component, making it easier for them to learn our extensions. It provides various analysis tools as well as workflow management capability. Moreover, Galaxy has not only made its source code available publically, but also hosts community space to encourage developers to build and share tools at http://usegalaxy.org/community.

The Web service-specific clients added provide an improved user experience over the Universal REST client, as the interface of the Universal REST client is limited to the features of a static tool in Galaxy. Apart from using information from the description documents of the Web services, the system also utilizes the information from the semantic annotations of their

parameters. Overall it provides a usable and efficient way of combining Web services and other bioinformatics analysis tools in workflows.

One drawback of the system is that it requires the users to know the location of the description documents for the various Web services, and assumes that the service providers provide good documentation of Web services. To eliminate this drawback a Web service discovery tool needs to be integrated into Galaxy, capable of suggesting useful Web services. Though lately, REST Web services are gaining popularity, SOAP is still the more established and used standard. Thus, the system needs be extended for use with SOAP Web services as well.

An example workflow built in Galaxy, using the operations provided by the EBI REST Web services is shown in Figure 9. This workflow demonstrates the ability to use Galaxy to perform BLAST with a given sequence and compare the alignment regions of highly similar sequences using ClustalW. The Web services used are WU-BLAST [35], dbfetch [36] and ClustalW2 [37]. The WADL documents for these Web services were created by us using the information provided on the EBI Web site. The first method used is run WU-BLAST. Two of the important input parameters of this method are the query sequence and the database to be searched. Other parameters are specified as well. This method returns an identifier for the job submitted to EBI. This job identifier is then fed to the getIds method that takes as input the job identifier and returns an array of result entry IDs of the hits in the BLAST response output. Next a tool that is already a part of Galaxy is used to convert the array to a comma delimited string, which is then fed to the fetchBatch method of dbfetch at EBI. This method fetches a set of entries in a defined format and style from a specified database. Next these entries serve as input for the runClustalW2 method, from ClustalW2 service at EBI, which submits a ClustalW2 job and returns its identifier. This identifier is given to the poll method of the ClustalW2 Web

service which waits for the job to finish and then gets the specified type of result data generated from the ClustalW job. The output of the poll method is converted into string format using a Galaxy tool byte2string.

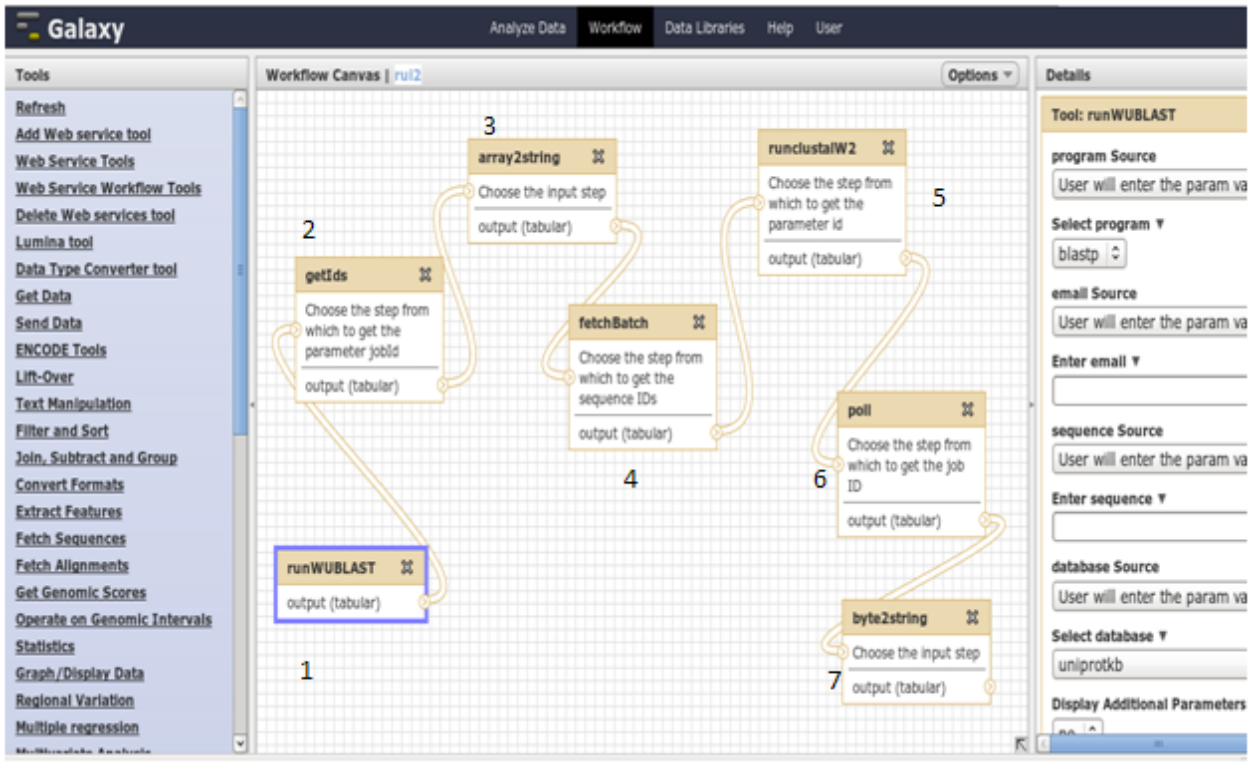

Figure 9: A workflow in Galaxy; with tools 1, 2, 4, 5, 6 representing Web services at EBI and tools 3 and 7 representing hypothetical Galaxy tools.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This chapter presents the conclusion drawn from this work and discusses some future work.

7.1. Conclusion

This thesis demonstrates a new way of extending Galaxy to support Web service functionality by introducing the Minimal Change approach, apart from extending the Restricted Tool Addition approach used by Shefali Shastri in her work [19].

We saw, that tools added via the Restricted Tool Addition approach are easy to maintain across versions of Galaxy, but lack a user-friendly interface, whereas tools added via the Minimal Change approach require changing the code base, but are much easier to use. The pros and cons of both the approaches and technicalities of their implementation were also discussed.

By incorporating support for semantically annotated REST Web services in Galaxy, we provide a skeleton of an expandable semantic Workflow Management System. In section 7.2 we propose some ideas to leverage the semantic extensions added to Galaxy.

7.2. Future Work

The work done can be extended in a number of ways. Below are some suggestions to further enhance the experience of biologists when connecting Galaxy to outside Web services.

Implementing these suggestions would require enhancing the current tools as well as adding new tools to Galaxy.

7.2.1 Service Suggestion and Discovery

The current extensions to Galaxy support semantic annotations by allowing addition of Web services described using SAWADL. Presently the semantic information is used only to generate tips about the input parameters. A natural extension of this provision would be using semantics to further enhance the user experience by incorporating service suggestion and discovery features. This would require addition of new tools for Web service suggestion and discovery. These tools would search for and suggest suitable Web services for an operation with a set of annotated input and output parameters.

Guidelines for a suggested interface of this tool in Galaxy have already been laid. It is called Lumina based on the original independent tool for Web service discovery provided by LSDIS lab at UGA [22].

7.2.2 Data Mediation

Another way to leverage the support for semantics in Web service tools is by adding data mediation functionality to Galaxy. A valuable contribution to the Web service tools would be the use of data mediation techniques to convert output of a tool into compatible input of another. These extensions would eliminate the need of any manual intervention for format conversion, making it much easier for the users to create workflows involving Web services.

7.2.3 Dynamic SOAP Clients

Though, support for SOAP Web services was added to Galaxy by Rui Wang [28] using the Maximal Functionality approach but, as discussed in section 3.2, these extensions cannot be easily integrated in newer versions of Galaxy. Using the Minimal Change approach we have presently added tools that enable only REST Web service support in Galaxy. However, the modular implementation of the Web service addition tool makes it extensible for SOAP Web services. The major difference being that instead of parsing a description document for REST Web service (WADL, WSDL 2.0 or SAWADL) it will require parsing a document describing a SOAP Web service (WSDL 1.1 or WSLD 2.0) to get the operation and parameter information. This information can then be used in the same way as in the current Web service addition tool to add SOAP Web service-specific clients as tools to Galaxy.

7.2.4 Delete Registered Services from GUI

The Minimal Change approach enables dynamic addition of tools via a GUI, a feature that is not inherent in Galaxy originally. Following this approach we can provide features to further enhance user experience like deleting Web service tools via the GUI. A user at some point might want to delete some of the previously added tools. Currently this requires editing the tool_conf.xml and restarting the server.

The refresh capability added to Galaxy in this project can be leveraged to provide support for dynamic tool deletion from the GUI. This requires development of a mechanism to map the tool name to its id, as Galaxy identifies the tools by their ids, but only the tool names are visible to the users. Issues concerning concurrent access and possible security breaches also need to be considered thoroughly.

REFERENCES

1. Blankenberg D, Von Kuster G, Coraor N, Ananda G, Lazarus R, Mangan M, Nekrutenko A, Taylor J. "Galaxy: a web-based genome analysis tool for experimentalists". Current Protocols in Molecular Biology. 2010 Jan; Chapter 19: Unit 19.10.1-21.
2. Blankenberg D, Taylor J, Schenk I, He J, Zhang Y, Ghent M, Veeraraghavan N, Albert I, Miller W, Makova K, Hardison RC, Nekrutenko A. "A framework for collaborative analysis of ENCODE data: Making large-scale analyses biologist-friendly". Genome Research. 2007 Jun; 17(6):960-4.
3. European Bioinformatics Institute 2010, an outstation of the European Molecular Biology Laboratory; information retrieved from http://www.ebi.ac.uk/.
4. DNA Data Bank of Japan; information retrieved from http://www.ddbj.nig.ac.jp/.
5. Kyoto Encyclopedia of Genes and Genomes, Kanehisa Laboratories; information retrieved from http://www.genome.jp/kegg/.
6. National Center for Biotechnology Information, U.S. National Library of Medicine; information retrieved from http://www.ncbi.nlm.nih.gov/.
7. Rodriguez-Tome, “The BioCatalog”, Bioinformatics, 14, pp. 469-470, 1998.
8. Fielding, Roy Thomas, “Architectural Styles and the Design of Network-based Software Architectures.”, Doctoral dissertation, University of California, Irvine, 2000.
9. Galaxy PSU; information retrieved from http://galaxy.psu.edu/.
10. Web Services Architecture, W3C Working Group Note, D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, 11 February 2004; available at http://www.w3.org/TR/ws-arch/.

11. Web Application Description Language, W3C Member Submission, Marc Hadley - Sun Microsystems, Inc, 31 August 2009; available at http://www.w3.org/ Submission/wadl/.
12. M. J. Hadley. Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006; available at https://wadl.dev.java.net/.
13. Web Services Description Language (WSDL) 1.1, W3C, E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. 15 March 2002; available at http://www.w3.org/TR/2001/NOTE-wsdl-20010315.
14. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C, R. Chinnici, J-J. Moreau, A. Ryman, S. Weerawarana, 26 June 2007; available at http://www.w3.org/TR/2007/REC-wsdl20-20070626
15. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C, M. Gudgin, et al., 24 June 2003, revised 27 April 2007; available at http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.
16. T. Berners-Lee, R. Fielding, L. Masinter. “Uniform Resource Identifiers (URI): Generic Syntax”. IETF RFC 2396, August 1998.
17. ApiDB: integrated resources for the apicomplexan bioinformatics resource center. 2007 Jan;35(NAR Database issue):D427-30. Aurrecoechea C, et al.
18. ApiDB (EuPathDB). GenesByTextSearch Web service; available at http://eupathdb.org/eupathdb/webservices/GeneQuestions/GenesByTextSearch.wadl.
19. Shefali Shastri. "Use of Semantics in Designing and Executing Scientific Workflows: A Case Study Using Galaxy". Masters Thesis (M.S. in CS Degree) December 2009, University Of Georgia.
20. Steve Menard. JPype homepage: http://jpype.sourceforge.net/.

21. Sean Bechhofer, Phillip Lord, Raphael Volz. “Cooking the Semantic Web with the OWL API”. 2nd International Semantic Web Conference, ISWC, Sanibel Island, Florida, October 2003. OWL API project homepage: http://owlapi.sourceforge.net/.
22. METEOR-S: Semantic Web Services and Processes, LSDIS and the University of Georgia; information retrieved from http://lsdis.cs.uga.edu/projects/meteor-s/.
23. Apache Woden: A WSDL 2.0 parser and validator. Woden homepage: http://ws.apache.org/woden/.
24. Jython project, homepage: http://www.jython.org/.
25. Mike Johnson, JEPP project, developed at Trinity Capital, homepage: http://jepp.sourceforge.net/.
26. Andi Vajda, JCC project, homepage: http://pypi.python.org/pypi/JCC.
27. OWL Web Ontology Language Overview, W3C, Deborah L. McGuinness, Frank van Harmelen, 10 February 2004; available at http://www.w3.org/TR/owl-features/.
28. Wang, R., D. Brewer, S. Shastri, S. Swayampakula, J. Miller, E. Kraemer and J. Kissinger. “Adapting the Galaxy Bioinformatics Tool to Support Semantic Web Service Composition". IEEE SWF 2009, IEEE ICWS 2009.
29. Lushbough, C., M. Bergman, C. Lawrence, D. Jennewein and V. Brendel. (2008). “BioExtract Server - An Integrated Workflow-enabling System to Access and Analyze Heterogeneous, Distributed Biomolecular Data”. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 99 (1).
30. BioExtract workflows; information retrieved from http://bioextract.org/help/about Workflows.html.

31. Oinn, T., M. Addis, J. Ferris, D. Marvin, M. Senger, T. Carver, M. Greenwood, K. Glover, M. Pocock, A. Wipat and P. Li. (2004). "Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics", 20 (7), 3045-3054.
32. Ludascher, B., I. Altintas, C. Berkley, D. Higgins, E. Jaegar, M. Jones, E. Lee, J. Tao and Y. Zhao. (2006). "Scientific workflow management and the Kepler system. CONCURRENCY AND COMPUTATION", 18 (10), 1039-1065.
33. Galaxy public site; information retrieved from http://main.g2.bx.psu.edu/.
34. Galaxy Add Tool Tutorial; information retrieved from http://bitbucket.org/galaxy/galaxy-central/wiki/AddToolTutorial.
35. European Bioinformatics Institute. (2009). WU-BLAST Web service; information retrieved from http://www.ebi.ac.uk/Tools/webservices/services/sss/wu_blast_rest.
36. European Bioinformatics Institute. (2009). dbfetch Web service; information retrieved http://www.ebi.ac.uk/Tools/webservices/services/dbfetch_rest.
37. European Bioinformatics Institute. (2009). ClustalW2 Web service; information retrieved from http://www.ebi.ac.uk/Tools/webservices/services/msa/clustalw2_rest.
38. 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.

APPENDIX A

INSTALLATION GUIDE

This guide assumes that Galaxy is installed on the user's system. If not, before going further with the installation of Galaxy REST Web service extensions download and install a local instance of Galaxy from http://bitbucket.org/galaxy/galaxy-dist/.

Prerequisites

i. JPYPE: Download and install JPYPE 0.5.4 from http://sourceforge.net/projects/jpype/files/.

ii. Python 2.6 : If not installed already, download and install Python 2.6 from http://www.python.org/download/. The extensions require Python 2.6 to be the default Python used by your Linux system.

iii. JRE : Download the appropriate Java for your system from the site : http://www.java.com/en/download/manual.jsp.

Once downloaded and installed, set the environment variable JAVA_HOME to point to the location of the installed version of Java, i.e., the directory containing *bin* and *lib* directories. Say, you installed JRE6 under /usr/java/ on your Linux machine. Use command

export JAVA_HOME=/usr/java/JRE6

Set GALAXY_HOME:

If, your source code is at location /user/galaxy/. Set an environment variable GALAXY_HOME to point to the location of the main Galaxy folder using the following command:

export GALAXY_HOME = /user/galaxy/

Universal REST client

Currently the support for the WADL, WSDL 2.0 and the SAWADL description documents are provided in separate REST clients that can be added as independent tools. In the future they will be provided as one REST client with support for all three specifications.

i. Download the tools:

All the universal clients can be found at http://cs.uga.edu/~guttula/ Galaxy/universalclients/.

Depending on your requirement download the tars for REST clients supporting SAWADL, WADL, and WSDL 2.0 from the following links:

REST client for WADL - http://cs.uga.edu/~guttula/Galaxy/universalclients/WADL/.

REST client for SAWADL - http://cs.uga.edu/~guttula/Galaxy/universalclients/ SAWADL/.

REST client for WSDL - http://cs.uga.edu/~guttula/Galaxy/universalclients/WSDL/.

ii. Unzip RESTclient:

Unzip the tars in the tools folder under the main Galaxy folder. This can be done as follows:

unzip ~/Downloads/ *tarname* –d GALAXY_HOME/tools/

This step should result in the tool folder to be present at GALAXY_HOME/tools/. So if you are installing all three clients you should have the tools at

GALAXY_HOME/tools/WADLRESTClient/

GALAXY_HOME/tools/SAWADLRESTClient/

GALAXY_HOME/tools/WSDLRESTClient/

iii. Modify tool_conf.xml:

Next step is to modify the tool_conf.xml file located at GALAXY_HOME. Add a section for each REST Web Service extension. This can be done for all three clients in the same way as shown below for the WADL client. Add the following lines after the last section tag is closed, but inside the tool tag in the GALAXY_HOME /tool_conf.xml.

<section name="REST client for WADL" id="rClient">

<tool file=" WADLRESTClient /WADLRESTclient1.xml" />

<tool file=" WADLRESTClient /WADLRESTclient2.xml" />

<tool file=" WADLRESTClient /WADLRESTclient3.xml" />

</section>

The installation for the universal clients is complete. When the user starts the server three new tools named REST client for WADL, REST client for SAWADL and REST client for WSDL should be visible in the left tool panel as shown in Figure 10.

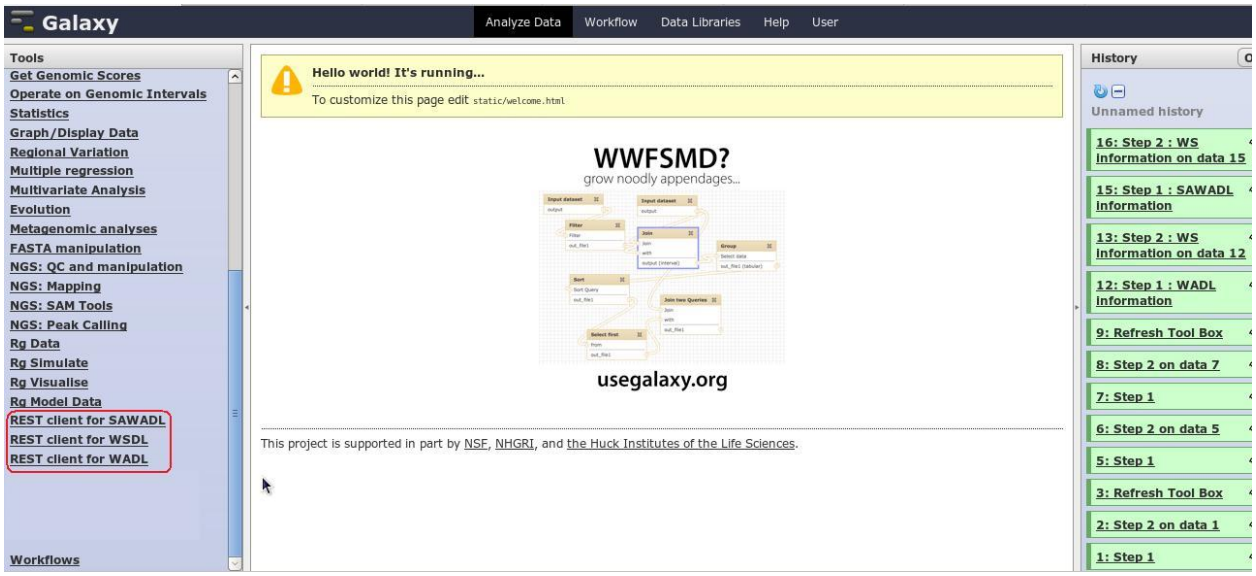

Figure 10: Galaxy toolbox with all the REST clients

REST Web service addition tool

i. Download the tool:

The REST Web service registration tool tar can be downloaded from http://cs.uga.edu/~guttula/Galaxy/WebServiceTool/.

ii. Unzip the tool:

Unzip the tar in the tools folder under the main Galaxy folder. This can be done as follows:

unzip ~/Downloads/ *tarname* –d GALAXY_HOME/tools/

This step should result in the "WebServiceTool" folder to be present under GALAXY_HOME/tools/.

iii. Modify Galaxy source code:

Two modules of Galaxy's source code need to be edited. Lines highlighted in bold *italics* are the code added to Galaxy's existing code.

a. lib/galaxy/tools/__init__.py

The Class Tool contains an execute method; edit it by making changes highlighted in *italics*, as shown in Table 1.

Table 1

```
def execute( self, trans, incoming={}, set_output_hid=True ):
    """
Execute the tool using parameter values in `incoming`. This just dispatches to the
`ToolAction` instance specified by 'self.tool_action'. In general this will create a `Job` that
when run will build the tool's outputs, e.g. `DefaultToolAction`.
    """
 if self.id == 'REFRESH_ID':
    self.app.refreshToolBox()
return self.tool_action.execute(self,trans,incoming=incoming,set_output_hid=set_output_
hid)
```

b. lib/galaxy/app.py

Add a new method called refreshToolBox to Class UniverseApplication in app.py.

The method to be added is shown in Table 2, in *italics*:

Table 2

```
def refreshToolBox( self):
    self.toolbox = tools.ToolBox( self.config.tool_config, self.config.tool_path, self )
```

iv. Modify tool_conf.xml:

Next step is to modify the tool_conf.xml file located at GALAXY_HOME. Add a section for the Web Service registration tool as shown below.

<section name = "Add Web service tool" id="RegisterWebServices">

<tool file = "WebServiceTool/WebServiceTool1.xml"/>

<tool file = "WebServiceTool/WebServiceTool2.xml"/>

</section>

Additional sections need to be added as placeholders for the clients that will be added using the above tool as follows:

<section name = "Web Service Tools" id="WebServices">

</section>

<section name = "Web Service Workflow Tools" id="WebServiceWorkflow">

</section>

NOTE: The above two sections should be added exactly as shown above, without any changes.

The installation for the dynamic Web service addition tool is complete. When the user starts the server three new sections named Add Web service tool, Web Service Tools and Web Service Workflow Tools should be visible in the left tool panel as shown in Figure 11.

Figure 11: Galaxy toolbox with dynamic Web service extensions

APPENDIX B

USER'S GUIDE

This guide assumes that the user has installed the Galaxy REST Web Service extensions on their system. For installation instructions check the Installation Guide in Appendix A. This document is intended to be a step by step guide through the functionality of the Galaxy REST Web Service clients added.

Universal Clients

Executing a REST Web service through the Universal REST client in Galaxy is a three step process. On clicking the REST client link in the Tools menu three steps can be seen. These steps have to be executed in order for the client to function properly. Figure 12 displays a screenshot of the Galaxy interface for "Step 1: WADL information". The tools added have been circled in red on the left tool panel.

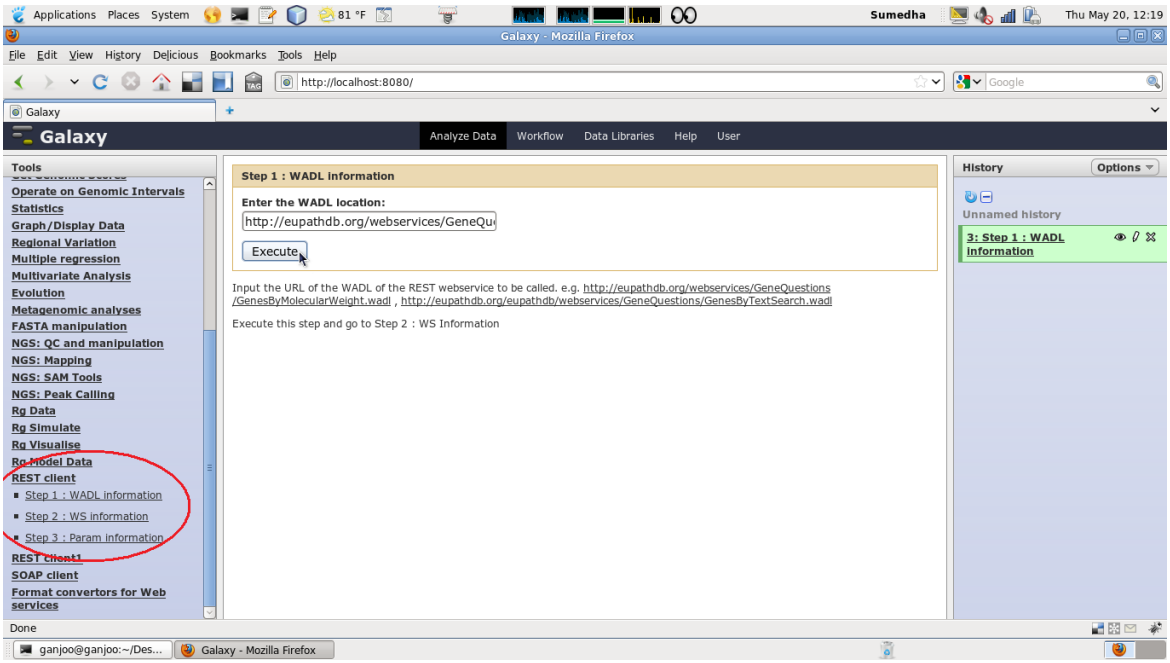

Figure 12: Step 1 of WADL REST client

Steps to follow:

i. Click on the first link “Step 1: WADL information” and enter the URL for the WADL, WSDL 2.0 or SAWADL document describing the REST Web service. Execute this step.

ii. Now move onto Step 2 by clicking on the second link “Step 2: WS Information”. Here you can see the result set generated by executing Step 1 selected as the first input. The # sign before a label indicates that the field is just for user’s information, and the user is not required to make any changes to it. Step 2 lets you choose the functionality offered by the Web service that you want to invoke and the resource you want to execute the functionality on. Once all the selections are made execute this step and go to Step 3. Figure 13 displays the interface of Step 2 of REST client in Galaxy.

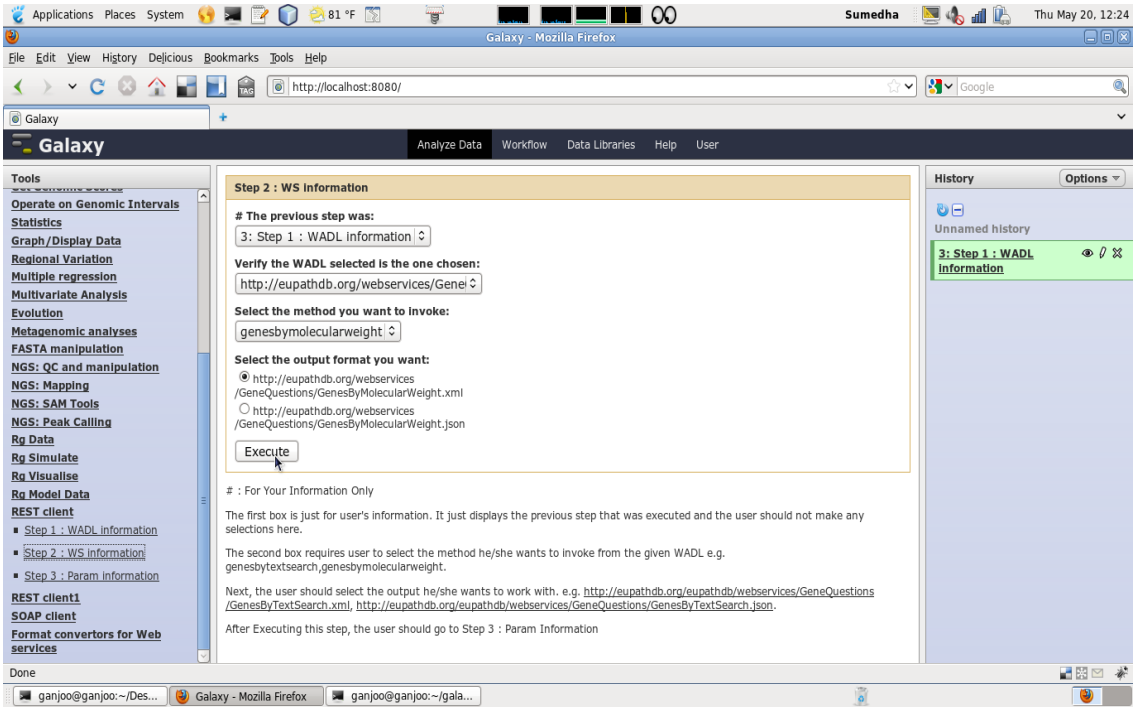

Figure 13: Step 2 of WADL REST client

iii. Now click on "Step 3: Param Information" to enter the parameter information. Information about the 0^{th} inputs parameter is displayed automatically. The following fields for Parameter 0 are displayed:

- Choose Input Parameter 0: This drop-down list displays the names of all parameters that the Web service allows the user to specify. Select a parameter you wish to enter a value for from this drop-down list.
- Enter Parameter value: This is a text-box for the actual parameter value to be entered into. Enter the input value corresponding to the parameter selected above.

  *Note: White spaces should be replaced with ** in the parameter value.*

Apart from the above fields, additional information is displayed about the parameter to help users in entering appropriate values.

- # Parameter Type is: This is to display the data type corresponding to the parameter selected. User does not have to make any selections here. This is provided merely for user's information.
- # Parameter Default Value is: This is to display the default value of the selected parameter. If no default value for the parameter is provided in the description document, it displays "None" as the default value of the parameter.
- #Whether Parameter is Required or not? : This either shows "true" or "false", the default being false. It displays "true" if the parameter selected is mandatory for invoking the Web service.

iv. The user can specify more parameters by clicking on the "Add new Input Parameters" button. It would display all the fields described above for each of the parameter selected. The above process can be repeated for all the input parameters of the Web service. Figure 14 shows Step 3 of the REST client.

v. Next, click on Execute to invoke the selected method of the REST Web service. The output from the Web service is displayed.

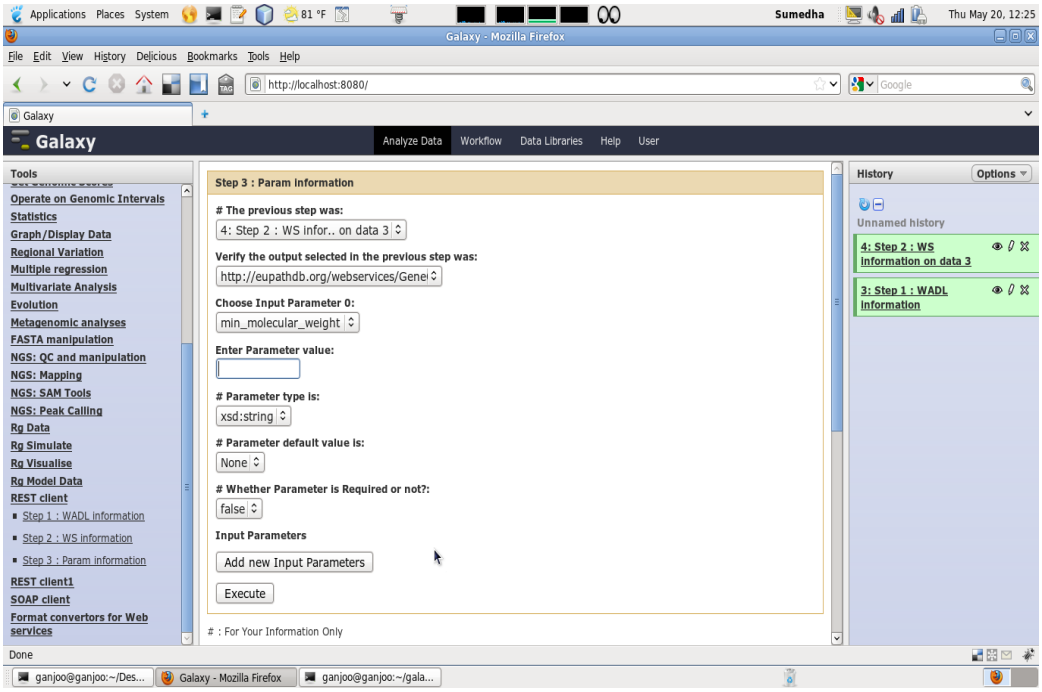

Figure 14: Step 3 of WADL REST client

Web Service-Specific Clients

The dynamic tool addition approach for invoking Web services requires users to not only add the Web Service Addition tool to Galaxy but also make changes in two modules of Galaxy's source code. This guide assumes that the user has the Web Service Addition tool installed and running on Galaxy before proceeding with this guide (Appendix A).

To use the Web service addition tool to register a REST Web service in Galaxy follow the instructions below:

i. Go to Tools on the left sidebar of the Galaxy screen and find "Step 1: Enter information about tool" under Add Web service tool. Enter the URL locating a WADL, WSDL 2.0 or SAWADL description document for the REST Web service you want to register, see Figure 15. Click on Execute and wait for the step to complete execution before going to "Step 2: Verify Settings".

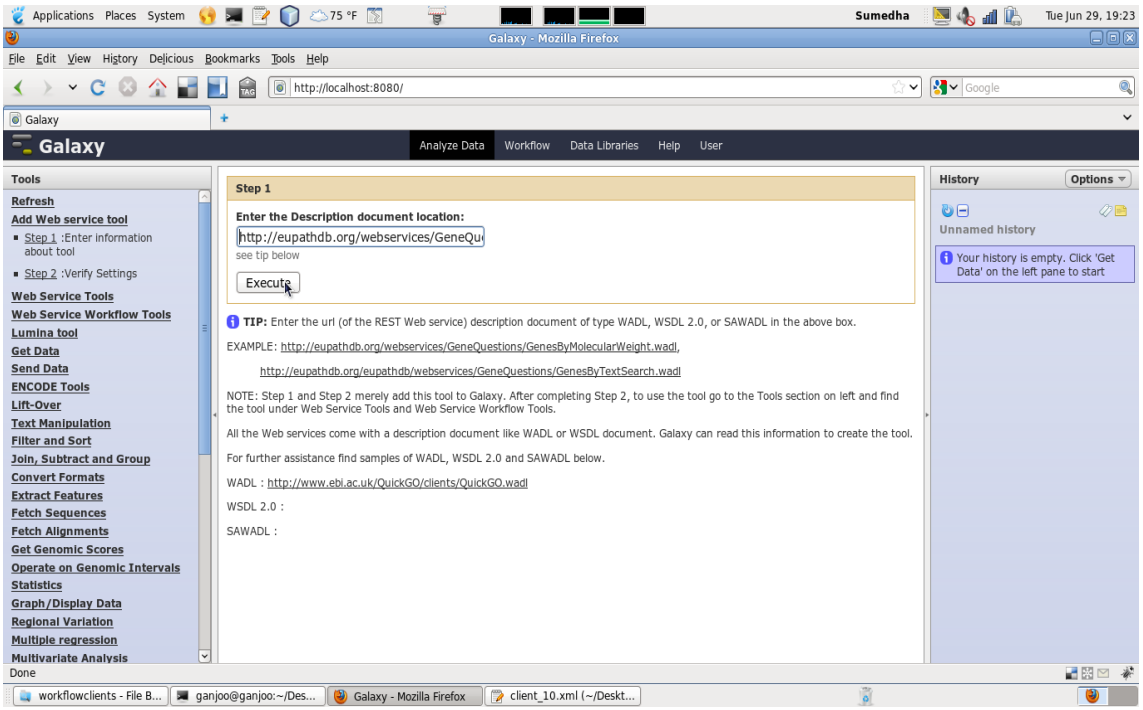

Figure 15: Step 1 of Add Web service tool

ii. In step 2, fields “a” and “b” are for your information only. No changes are to be made to them. Fields “c” and “d” let you refine the tool settings. Choose the functionality you want to invoke and the resource you want to invoke it on. See an example usage in Figure 16. After executing step 2, go to “Step 3: Refresh Galaxy”.

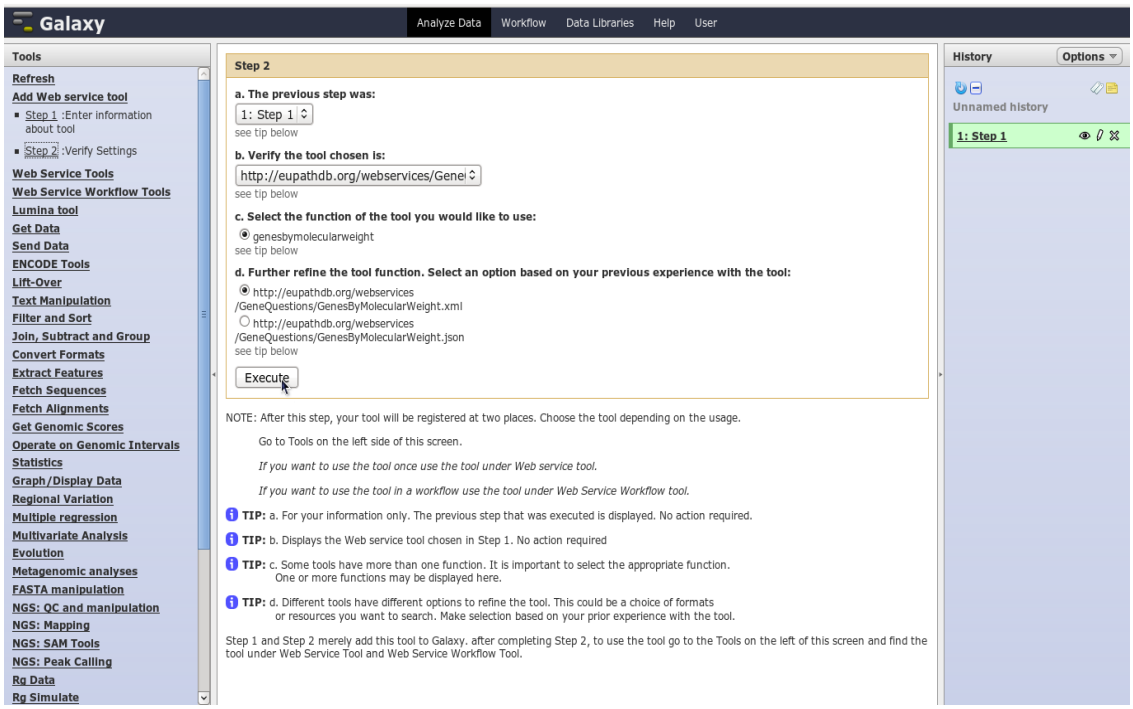

Figure 16: Step 2 of Add Web service tool

iii. Step 3 does not require any input from you and merely requires you to execute it to refresh Galaxy’s toolbox.

Step 1 and 2 add the tool to Galaxy and Step 3 refreshes Galaxy to load the newly added tool. The tool added should now be visible under “Web Service Tools” and “Web Service Workflow Tools”. If not visible, click on the Galaxy icon on the top left of the screen.

For one-time invocation of a Web service use the tool added under “Web Service Tools”, for use in workflows go to the tool added under “Web Service Workflow Tools”. Next we explain the working of the tool added under “Web Service Tools” with the help of an example. The REST Web service in the example is “genesbytextsearch” from EuPathDB, a WADL for which can be found at http://eupathdb.org/eupathdb/webservices/GeneQuestions/GenesByTextSearch.wadl.

On clicking on genesbytextsearch under Web Service Tools you will see an interface to enter parameter values as shown in Figure 17. Only required parameters are displayed, with an option to display the rest. Depending on the parameter either a drop-down or a text box is available to let the user specify its value. At the bottom tool tips providing information like data type about each parameter exist. On executing this tool the *genesbytextsearch* service is invoked and the output is displayed.

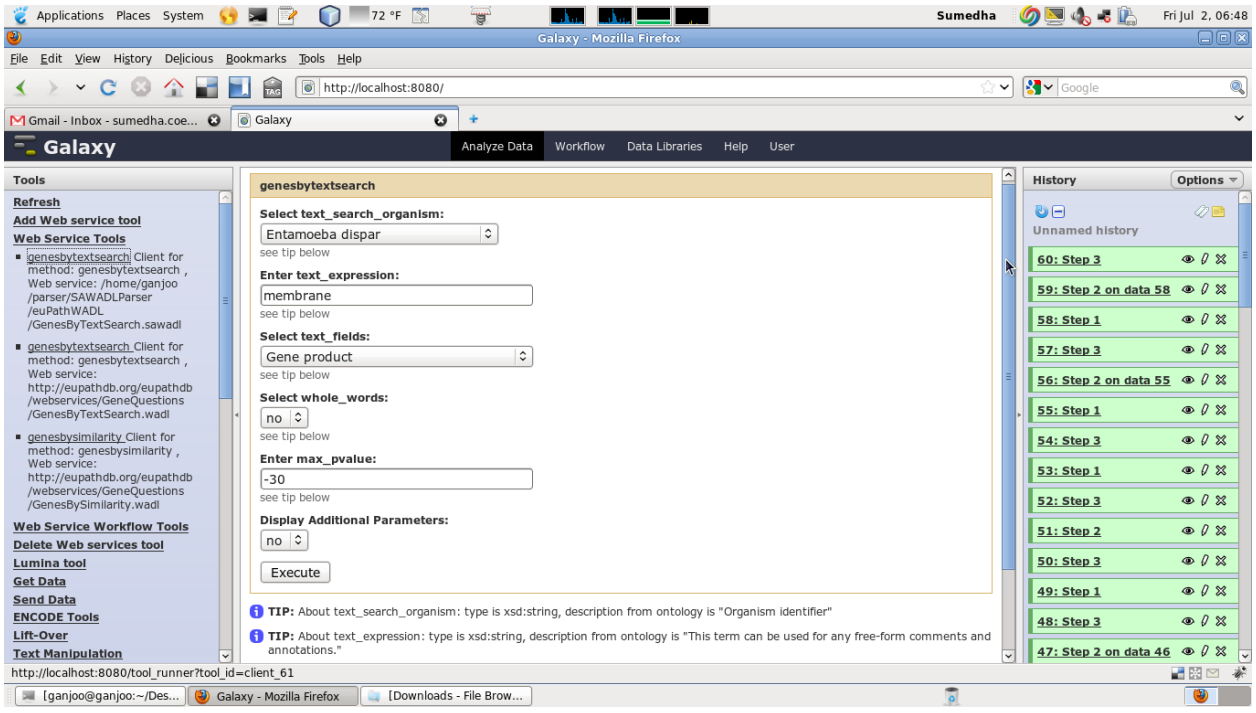

Figure 17: genesbytextsearch [18] Web service-specific client in Galaxy

APPENDIX C

DEVELOPER'S GUIDE

This developer's guide provides a brief overview of the work done to implement the REST clients to support WADL, WSDL and SAWADL parsing. It is intended to aid developers in understanding how the clients work with the larger aim of demonstrating how to further enhance the Web service support and REST clients in Galaxy. It is assumed that the developer has set up an instance of the Galaxy server and the Galaxy REST Web service extensions as instructed in the Installation Guide. For further information a tutorial on adding tools to Galaxy can be found at http://bitbucket.org/galaxy/galaxy-central/wiki/AddToolTutorial

Universal REST clients

If you have followed the instructions in the Installation Guide the environment variable GALAXY_HOME of your system would point to the location of the main Galaxy folder. The REST Web service extensions to Galaxy are added as tools to Galaxy. The various universal clients added as tools to Galaxy now should be found at:

GALAXY_HOME/tools/WADLRESTClient/

GALAXY_HOME/tools/SAWADLRESTClient/

GALAXY_HOME/tools/WSDLRESTClient/

Hence, the source code for an added universal client, say WADL client can be found at GALAXY_HOME/tools/ WADLRESTClient. This represents the root folder for the tool. For each universal client under its root folder following files and folders can be found:

i. eXtended Markup Language (XML) files: Each XML file is used to describe the interface for one step in the REST client. Every step is treated as a separate tool here. In our case all REST clients have been implemented in three steps and thus, would have three XML files. To learn about various tags used in Galaxy tool XML files refer to http://bitbucket.org/galaxy/galaxy-central/wiki/ToolConfigSyntax.

ii. Python (.py) files: The REST clients have been implemented so that each tool(step) invokes Python code implemented in a separate Python (.py) file. Thus, we have three Python files per client. The function and implementation of each is discussed later.

iii. lib folder: Apart from the .py and .xml files, the REST client may contain a folder named lib. This folder contains all jars and Java classes required by the clients for WADL, WSDL 2.0 or SAWADL parsing.

Three universal REST clients are added: WADL REST client, WSDL 2.0 REST client, SAWADL REST client. Below is a comparison of the three clients and some more details about each.

Table 3: Comparison of WADL, WSDL and SAWADL REST clients

	WADL REST client	WSDL REST client	SAWADL REST client
Description Language	WADL	WSDL 2.0	SAWADL
Location	GALAXY_HOME/tools/ WADLrestClient/	GALAXY_HOME/tools/ WSDLrestClient/	GALAXY_HOME/tools/ SAWADLrestClient/
XML files	3 XML files: 1.WADLrestClient1.xml 2.WADLrestClient2.xml	3 XML files: 1.WSDL2restClient1.xml 2.WSDL2restClient2.xml	3 XML files: 1.SAWADLrestClient1.xml 2.SAWADLrestClient2.xml

	3.WADLrestClient3.xml	3.WSDL2restClient3.xml	3.SAWADLrestClient3.xml
.py files	3 Python files: 1.WADLrestClient1.py 2.WADLrestClient2.py 3.WADLrestClient3.py	3 Python files: 1.WSDL2restClient1.py 2.WSDL2restClient2.py 3.WSDL2restClient3.py	3 Python files: 1.SAWADLrestClient1.py 2.SAWADLrestClient2.py 3.SAWADLrestClient3.py
Java classes/ jars in lib	LSDIS WADL Parser	Apache Woden Parser	LSDIS SAWADL Parser, OWL-API

These REST Clients are universal clients that can invoke any REST Web service given a description document for that Web service. All REST clients follow a three step process to invoke any Web service. The Python modules and XML files for the WADL REST client is described below.

WADL REST client

Note: All the Python modules in this client use WADL Parser made available by LSDIS lab, UGA to parse the WADL file and obtain the required information at each step. The WADL Parser is written in Java and is invoked from Python using JPype.

WADLrestClient1.xml

This XML file generates the interface in Galaxy for the first step of WADL REST client. It generates a tool which lets the user either input the URL of the WADL describing the REST Web service. This tool invokes WADLrestClient1.py, described below, on executing. Figure 18 shows the WADLrestClient1.xml with an explanation of the various tags used.

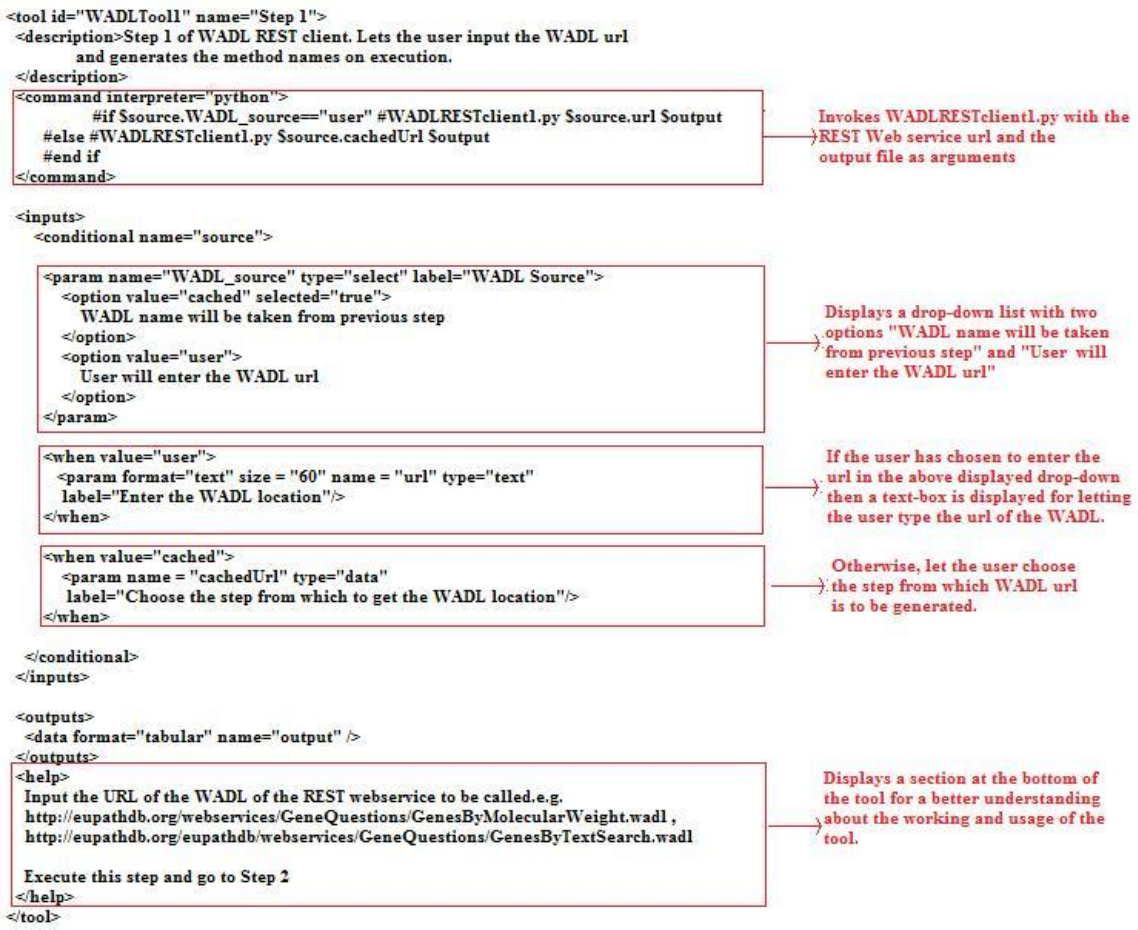

Figure 18: Parts of WADLRESTClient1.xml

WADLrestClient1.py

This module is the first step in the WADL REST client. Depending on the client this tool takes the URL of the file describing the Web service i.e, the WADL file as input. It parses the WADL file to get the methods provided by the REST Web service. It then writes a local file containing the operations of the Web service and the resources that support each operation.

WADLrestClient2.xml

This XML file generates the interface in Galaxy for the second step of WADL REST client. It generates a tool which lets the user select the resource format and the method to invoke

from the previously specified REST Web service. The user is also supposed to verify that the first select box shows the previous step to be Step 1 and the WADL selected is the one specified in the previous step. This tool invokes WADLrestClient2.py, described below, on executing. Figure 19 shows a part of WADLrestClient2.xml with an explanation of some of the important tags used.

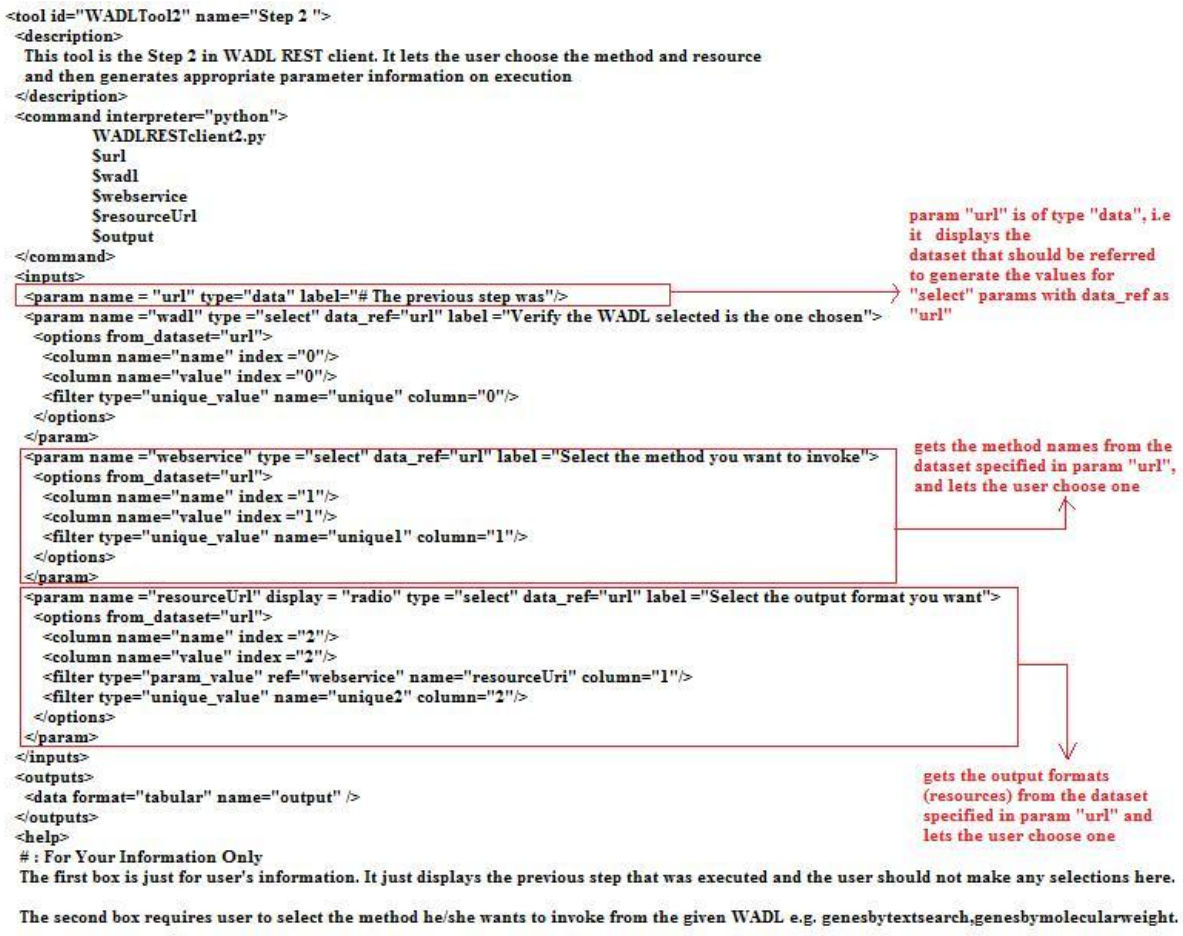

Figure 19: Parts of WADLRESTClient2.xml

WADLrestClient2.py

This module is the second step in the WADL REST client. It takes the WADL location, the method and the resource selected by the user as input. It parses the WADL to obtain the

parameter information about the method selected. It then writes a local file containing all the parameters, with information about their data type, default behavior and if they are required or not.

WADLrestClient3.xml

This XML file generates the interface in Galaxy for the third step of WADL REST client. It generates a tool which lets the user choose the parameters and enter their values. Also help is provided by displaying the parameter type, default value and if the parameter is required or not. Again the user is also supposed to verify that the first select box shows the previous step to be Step 2 and the WADL selected is the one specified in the previous step.

On executing this tool invokes WADLrestClient3.py. Figure 20 shows a part of WADLrestClient3.xml with an explanation of some significant tags used.

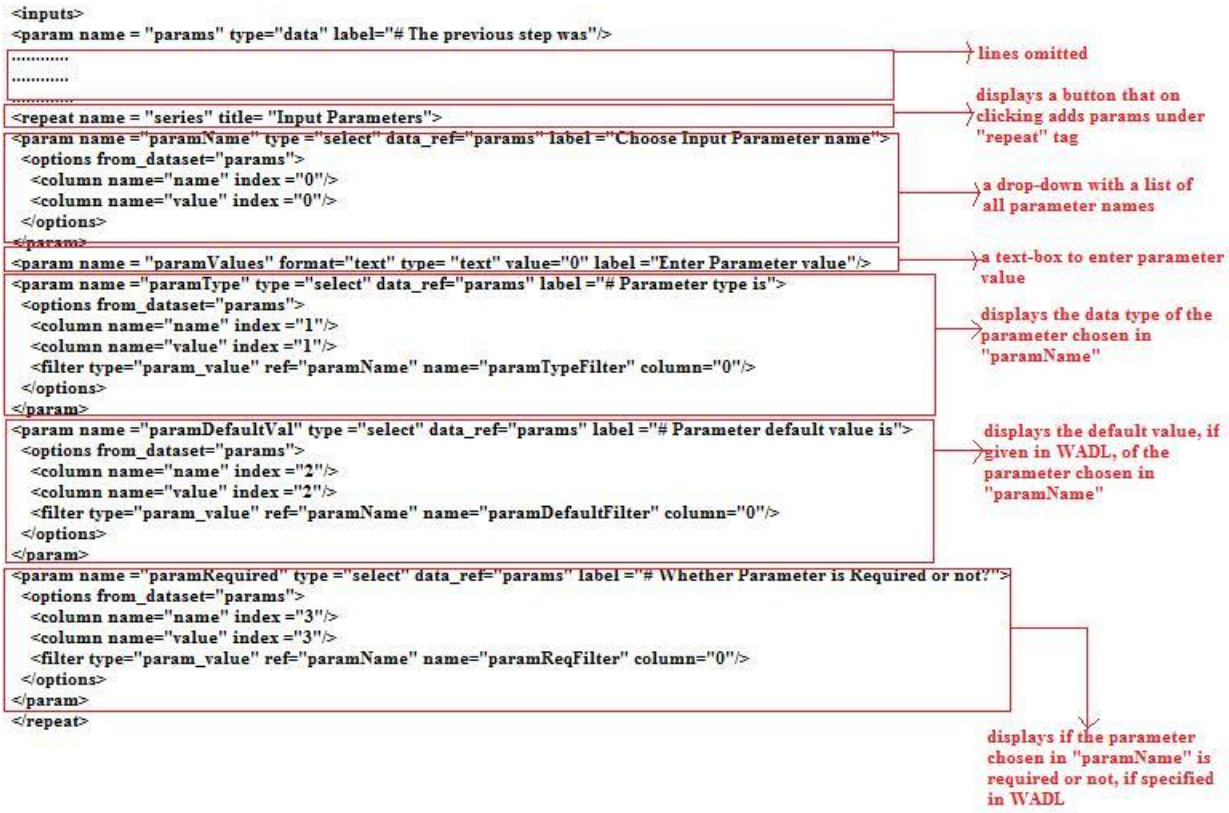

Figure 20: Parts of WADLRESTClient3.xml

WADLrestClient3.py

This module is the third step in the WADL REST client. In every parameter value passed to this module a white space character is replaced by a double quote (**) character. In this module this ** is replaced back by white space. Also it gets rid of any trailing or preceding white spaces in the parameter values. Then it takes the parameter names and value pairs and creates a dictionary of the type {parameter name: parameter value}. After URL encoding the dictionary, the REST Web service is invoked by passing the resource URL and the parameter dictionary to the urllib.urlopen method. The results obtained from invoking the service are written to a local file.

WSDL REST client

WSDL REST Client is similar to WADL REST Client in its behavior and file structure. The main difference in the two clients is that it invokes a REST Web service described using a WSDL 2.0 document, instead of a WADL document. The Apache Woden parser is used to parse the WSDL 2.0 document. In WADLrestClient1.py given the WADL URL all methods defined in it were written to the output file, whereas here in WSDLrestClient1.py only services with HTTP binding are written to the output file. Figure 21 shows an example of a part of a WSDL 2.0 document describing a service with a HTTP binding.

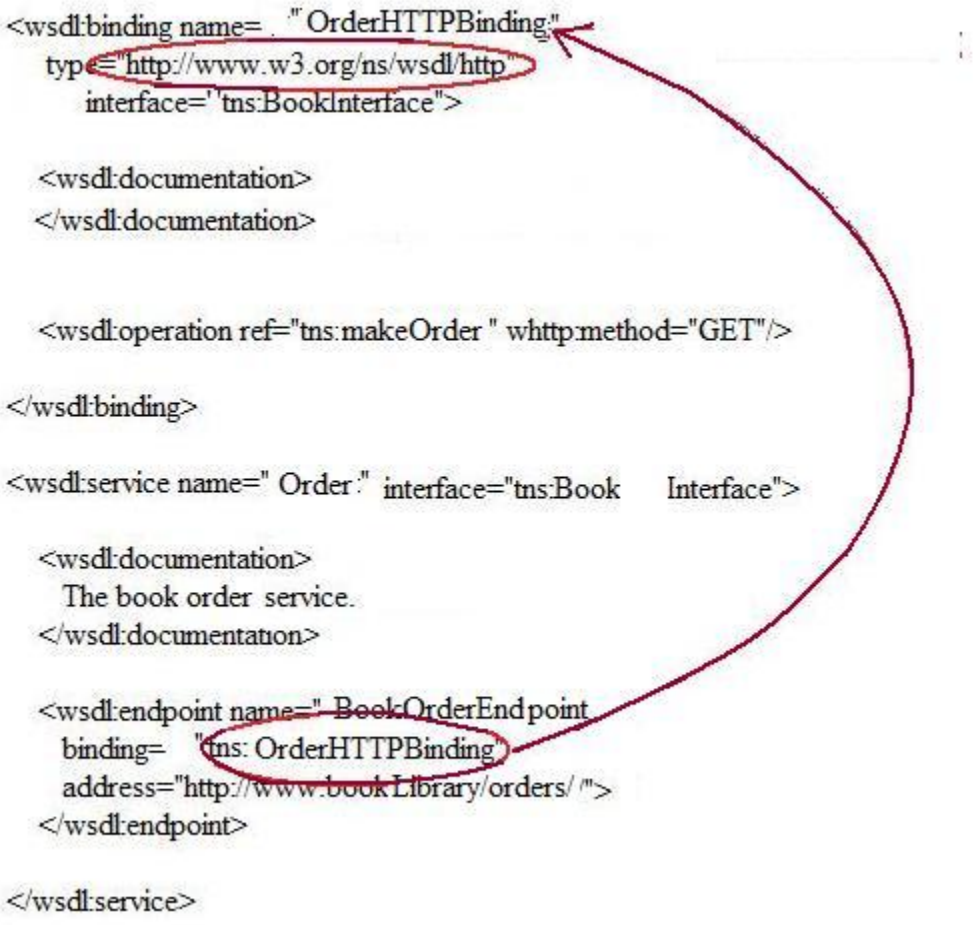

Figure 21: Example of HTTP binding in WSDL 2.0

SAWADL REST client

SAWADL REST client has the same file organization and naming conventions as the WADL REST client. Instead of using a WADL for describing the REST Web service, this client deals with semantically annotated services that are described using a SAWADL document.

The first two steps work exactly the same way as the WADL client's, the only difference being that the parser used here is the SAWADL Parser from the LSDIS lab, UGA, instead of the WADL Parser. In Step 3, like the WADL REST client Step 3 the parameter names are displayed in a drop-down list. The main difference in implementation is the introduction of the "Help" feature, which requires user to select the "Display Help" option from the "Help" drop-down list to show additional information about each parameter. The additional information also contains information obtained from the annotation of the parameter. In SAWADLrestClient2.py for each parameter the model reference attribute is checked in SAWADL. If the parameter is annotated with a valid concept from an ontology, it reads the ontology and gets the "comment", if any, about the concept from the ontology. OWL-API is used for parsing of ontology and accessing the comment attribute value of the concepts.

Web Service-specific clients

Galaxy does not support dynamic addition of tools by itself. To enable addition of Web service clients as tools dynamically via the GUI, we follow an approach that requires minimal changing of Galaxy source code and addition of an external tool.

The source code is changed to support reloading of the toolbox without restarting the server, enabling refreshing of the "Tools" section in the left side panel of Galaxy GUI. This requires changing two modules of the Galaxy source code.

i. lib/galaxy/app.py: app.py module defines a class named UniverseApplication. We add a new method to this class called refreshToolBox. This method creates a new instance of Toolbox given a tool_conf.xml file and its path and assigns it to the instance variable toolbox of the current instance of UniverseApplication.

ii. lib/galaxy/tools/__init__.py: One of the classes defined in this module is Tool. It represents a tool that can be embedded in Galaxy. The method named execute defined in the Tool class gets executed whenever a tool is executed through the GUI. We embed code in this method to invoke refreshToolBox method on the instance of UniverseApplication class whenever the tool with id REFRESH_ID is executed.

A summary of the changes to Galaxy source code is shown in Figure 22. These changes incorporate the ability to refresh Galaxy toolbox without having to restart the server. We leverage this feature to add the ability to add dynamic Web service clients as tools via GUI.

Figure 22: Changes to Galaxy source code

The Web Service Addition Tool has three steps. The first is described using WebServiceTool1.xml. On executing step 1 WebServiceTool1.xml invokes WebServiceTool1.py passing it the description document URL entered by the user. WebServiceTool1.py checks whether the document is of type WADL, WSDL or SAWADL. Depending on the document type it calls the method of class Document from getMethods.py. This method parses the description document and outputs the method information from it. This information is interpreted by the WebServiceTool2.xml enabling step 2 of the tool to let user make selections to refine the tool functionality to be added. On executing step 2

WebServiceTool2.py is invoked which initializes an instance of ClientGenerator class from generateClient.py and an instance of ClientGenerator1 class from generateClient1.py. ClientGenerator and ClientGenerator1 classes define methods to create clients for WADL, WSDL 2.0 and SAWADL, called wadlClient(), wsdlClient() and sawadlClient() respectively. These methods in ClientGenerator parse the description document and create a client, in XML, specific to the Web service method specified. This client invokes a specific REST Web service with the values given for the required parameters. It can also optionally take values for other parameters as input. Each input parameter of the Web service is either represented as a text-box or a drop-down list. If the parameter has a list of options specified for the values it can assume, it is represented with a drop-down list otherwise as a text-box. The default value is also read and displayed in both the cases. This tool is added in the folder named clients and is displayed under Web Service Tools in the GUI.

The methods of ClientGenerator1 also add a Web service-specific client, but under the folder named workflowclients. These clients are displayed under Web Service Workflow Tools in the GUI. The only difference being that this client can take the input for each parameter from a previous file as well as from the user. Thus, this client can be used as a part of workflows in Galaxy. After both the clients are generated, WebServiceTool2.py invokes edit_tool_conf.py to add to the tool_conf.xml file the location of the newly generated clients at appropriate places. This enables Galaxy to recognize the newly added clients as Galaxy tools. The last step, Step 3, reloads the toolbox, enabling Galaxy to identify the newly added tools without having to restart the server. Figure 23 describes the working of the Web Service Addition Tool with the help of a diagram.

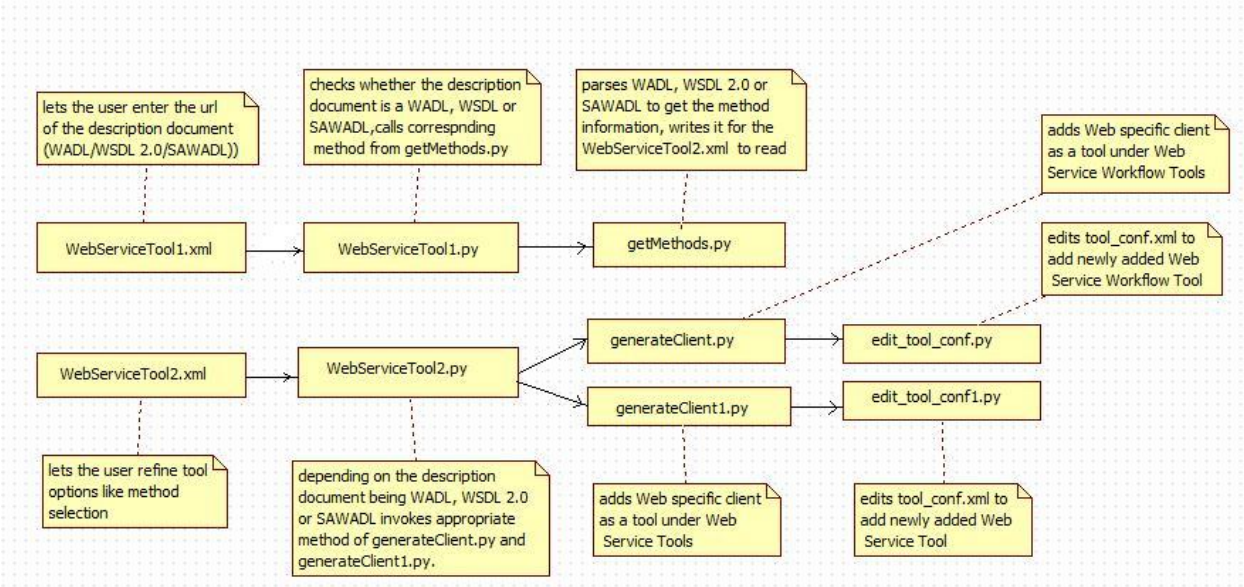

Figure 23: Interaction of added files for Web service addition and invocation