

USE OF SEMANTICS IN DESIGNING AND EXECUTING SCIENTIFIC WORKFLOWS:

A CASE STUDY USING GALAXY

by

SHEFALI SHASTRI

(Under the Direction of John Miller)

ABSTRACT

Galaxy is a bioinformatics framework which contains a number of tools to perform analysis on biological data. There exist a huge number of Web services for bioinformatics. However, at present Galaxy does not have any support for Web services. The focus of this thesis is to add Web service capability to Galaxy. For this purpose we have created a universal REST client and a universal SOAP client and added them as tools to Galaxy. The thesis discusses these two clients in detail. It also discusses how these clients can be used with other Galaxy tools in a workflow. It also presents multiple ways of utilizing semantic annotations to facilitate discovery, composition and user interaction with Web services and processes/workflows.

INDEX WORDS: Galaxy, bioinformatics Web services, REST, SOAP, Semantics, SAWSDL, WSDL-S, WADL-S

USE OF SEMANTICS IN DESIGNING AND EXECUTING SCIENTIFIC WORKFLOWS:

A CASE STUDY USING GALAXY

by

SHEFALI SHASTRI

B. E., University of Mumbai, India, 2006

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

2009

© 2009

Shefali Shastri

All Rights Reserved

USE OF SEMANTICS IN DESIGNING AND EXECUTING SCIENTIFIC WORKFLOWS:

A CASE STUDY USING GALAXY

by

SHEFALI SHASTRI

Major Professor: John Miller

Committee: Budak Arpinar
Eileen Kraemer

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2009

DEDICATION

To my parents, brother and my husband, Chinmay.

ACKNOWLEDGEMENTS

I would like to thank Dr. John Miller, my advisor, for all the guidance and support provided to me in the course of this thesis and the one and a half years I spent at the LSDIS lab. My colleagues Rui Wang and Srikalyan Swayampakula have always offered help whenever I needed it. I thank them for their support and encouragement. Lastly I would like to thank my family and friends for giving me much needed morale boosts from time to time.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	x
CHAPTER	
1 INTRODUCTION	1
2 SURVEY OF SCIENTIFIC WORKFLOW TOOLS	3
2.1 Usability	3
2.2 Support for Web Services	5
2.3 Potential Use of Semantics.....	6
2.4 Introduction to Galaxy.....	7
3 WORKFLOWS IN GALAXY.....	9
3.1 Workflows from Scratch	9
3.2 Workflows from History	10
4 INVOKING WEB SERVICES THROUGH GALAXY	15
4.1 Approaches to Integrating Web Service Functionality	15
4.2 Background on Web Services	16
4.3 Invoking SOAP Services.....	17
4.4 Invoking RESTful Services.....	24
5 UTILIZING SEMANTICS FOR GALAXY WORKFLOW	28
5.1 Utilizing Semantics in Workflow Design	28

5.2 Semantic Annotations to WADL Documents	29
5.3 Additional Semantics or Data and Process Mediation	33
6 APPLICATION SCENARIOS	35
6.1 Workflow Using SOAP Client on EBI Web Service	35
6.2 Workflow Using REST Client on EBI Web Service	36
7 EVALUATION.....	39
8 CONCLUSIONS AND FUTURE WORK	41
REFERENCES	42
APPENDICES	44
A GALAXY WEB SERVICE EXTENSION USER'S GUIDE	44
B GALAXY WEB SERVICE EXTENSION DEVELOPER'S GUIDE	49

LIST OF FIGURES

	Page
Figure 3.1: An Example Workflow in Galaxy.....	10
Figure 3.2: Data from EuPathDB.....	11
Figure 3.3: Filtering Only Those Genes Which Have Description.....	11
Figure 3.4: Cutting Columns C1 and C3 from the Data	12
Figure 3.5: Output Showing the Gene and Description Columns Only	12
Figure 3.6: Choose Which Tools You Need In the Workflow	13
Figure 3.7: Workflow Created From History	13
Figure 4.1: SOAP Web Service Client Extension to Galaxy.....	18
Figure 4.2: Enter the URL of the WSDL.....	18
Figure 4.3: Output of the First Step of the SOAP Client	19
Figure 4.4: Operations of the Web Service.....	20
Figure 4.5: Output of the Second Step of the SOAP Client.....	20
Figure 4.6: Invoking the Web Service	22
Figure 4.7: Output of the Chosen Operation.....	23
Figure 4.8: Format Convertors for Web Services Toolset.....	23
Figure 4.9: File Saved In Proper FASTA Format.....	24
Figure 4.10: Restful Web Service Client Extension to Galaxy	25
Figure 4.11: Inputs to Restful Web Service Through Universal REST Client.....	26
Figure 4.12: Output of the Universal REST Client.....	27

Figure 5.1: Sample WADL Document	30
Figure 5.2: Meta-Model For WADL-S	33
Figure 6.1: Edited Workflow with Two Additional Tools	36
Figure 6.2: Output of the SOAP Workflow	36
Figure 6.3: Workflow Using REST Client and FASTA Manipulation Tools	37
Figure 6.4: Output of Step3 and Step4 of The Workflow.....	38

LIST OF TABLES

Table 5.1: Proposed WADL-S Semantic Annotation Attributes32

CHAPTER 1

INTRODUCTION

Galaxy (Taylor, 2007) is a bioinformatics framework which provides various tools for performing analyses on different types of bioinformatics data. Developers can create their own tools to add them to their local instance of Galaxy. There exist many Web services for bioinformatics analysis on the Internet. It would be easier for users to use these readymade Web services rather than create a tool in Galaxy that performs exactly the same actions. However, at present Galaxy does not have provision for executing Web services or adding them to the existing workflow system. This thesis focuses on adding both REpresentational State Transfer (REST) and SOAP¹ Web service capability to Galaxy. Two universal clients for each kind of Web service are added as tools to Galaxy. Using these tools a Web service can be executed both individually as well as a part of a workflow in Galaxy. Since the users of Galaxy are biologists, it is essential to keep the user interface simple and easy to use. Keeping that goal in mind it is further discussed how semantic annotations to Web services can make it easier for the user to use these clients. A preliminary design for a semantically annotated Web Application Description Language (WADL-S) is also discussed.

The thesis is organized as follows. Chapter 2 contains a survey of contemporary scientific workflow tools popular in the bioinformatics area. Chapter 3 will explain how workflows are

¹ Formerly an acronym for Simple Object Access Protocol

designed in Galaxy. In Chapter 4, a brief background on Web services will be given. This will be followed by the description of how SOAP Web services as well as RESTful Web services can be executed through Galaxy. Chapter 5 will discuss utilizing semantics in workflow design and workflow execution. Chapter 6 describes two scenarios where the SOAP and REST extensions to Galaxy have been used. Chapter 7 gives a brief evaluation of the SOAP and REST extensions. In Chapter 8, conclusions and future work will be discussed.

CHAPTER 2

SURVEY OF SCIENTIFIC WORKFLOW TOOLS

In the last decade, due to advances in computing, scientific experiments have become more evolved. As a result there is huge amount of scientific data generated and analytical tools are required to make sense of this data. Since frequently the same set of analyses may have to be performed on different sets of data the process can get repetitive. This is where scientific workflows help. A number of repetitive interdependent tasks can be stored as a workflow and this workflow then can be executed on different sets of data.

There exist a number of frameworks which help scientists analyze their data with different tools and create workflows. Galaxy is one such framework which enjoys substantial popularity with the bioinformatics community.

In this chapter, we will take a look at three such scientific workflow tools and evaluate them on the basis of usability, support for Web services and potential use of semantics. The three tools are Taverna (Oinn, 2004), Kepler (Ludascher, 2006) and BioExtract (Lushbough, 2008).

2.1 Usability

Scientists are generally not seasoned programmers and hence the user interface that the workflow tools provide should allow the user to construct, edit and execute workflows but hide the complexity behind these tasks.

Taverna has a simple uncluttered interface which lets the user see the different Web services and data sources available, the currently constructed workflow as well as the services included in the current workflow, all in the same view. The view of the workflow can be adapted to observe different aspects, to show all the ports for all the services, only those ports that have been connected or bound, or to change the layout of the workflow from portrait to landscape (Fisher, 2009). While the workflow is executing, it shows the progress of the workflow through color codes. The Web services that have finished executing are green, ones currently being executed are purple and those awaiting execution are grey.

In Kepler, workflows are built from actors that perform computational tasks (McPhillips, 2009). Each of these actors is a component in the workflow and performs specific tasks. Kepler makes a distinction between the graphical representation of a workflow and the model of computation used to execute it. This means that different models of computation can be associated with the same workflow graph. Kepler inherits several common models of computation from the Ptolemy (Lig, 1999) system, including Synchronous Data Flow (SDF), Continuous Time (CT), Process Network (PN), and Dynamic Data Flow (DDF), etc. Thus, a point of control is required that will specify the model of computation to interpret and execute a workflow. This point of control is called a director in Kepler and has to be specified by the workflow author. Kepler also provides a search tool that can search for actors based on their functionality.

Kepler makes scientific workflow creation simple, users can drag and drop components on the workflow creation area, connect these components to indicate data flow. “Kepler represents the overall workflow visually so that it is easy to understand how data flow from one component to another. The resulting workflow can be saved in a text format, emailed to colleagues, and/or published for sharing with colleagues worldwide” (Ludascher, 2006).

BioExtract Server is a web-based system. This means that users do not have a local instance of the system. They have to go to <http://bioextract.org/> and create a user account on the BioExtract system. Once the user is logged in he/she can create new workflows, upload existing workflows from their local system, edit existing workflows and share workflows. BioExtract records tasks performed by the user on the system and saves this into a workflow. If these are tasks that a user needs to perform repetitively then this saved workflow can be used to automate the process. This is convenient, however, the disadvantage of this feature is that if a certain task has failed during execution, the recorder does not realize this and includes it as a step in the workflow. Though the user can delete steps from the workflow this can be cumbersome if the workflow has a large number of steps. The whole workflow can be executed as it is or individual steps can be executed separately. Execution can also be paused to allow the user to check progress, or adjust input into a step. Steps in the workflow can be modified. This allows the user to repeat the workflow on different data, or with modified parameters (BioExtract Server, 2008). Workflows created by BioExtract are always sequential. Also, there is no way to add a step or change the order of steps in a stored workflow.

2.2 Support For Web Services

“Web services allow applications to communicate across platforms and programming languages” (OASIS, 2009). More importantly, accessing them is completely independent of the language in which they are coded. It is hence necessary that these frameworks have some Web service functionality.

Taverna has good support of Web services. Not only does it have an extensive registry of Web services but it also lets the user add external Web services without undue effort. Often users will need to write snippets of Java code that will map inputs and outputs. The Web service support is extended only towards SOAP Web services as of now. There is as yet no way to add RESTful Web services to Taverna.

Kepler also provides support for Web services. Web services can be invoked using the Web service actor in Kepler. This actor accepts the WSDL location, the method name to invoke and input parameters from the user. Kepler too has no support for RESTful Web services.

BioExtract too provides a list of Web services through its registry to the user. These Web services can be searched on keywords. BioExtract though provides no functionality to access external Web services. Hence, the user can only use the Web services listed in BioExtract.

2.3 Potential Use Of Semantics

Semantic annotations add meaning to interface descriptions of software programs. During workflow compilation it is important to know whether two interdependent tools are compatible syntactically and semantically, since one does not guarantee the other. Also, semantic annotations help during Web service discovery and composition.

Taverna itself does not have any provisions for semantic Web services but it can use the Feta Plug-in to discover Web services on basis of semantics. In Taverna, services can be discovered by exploiting their semantic descriptions. The semantic search works only on services registered the Feta repository.

The design of Kepler lets users register their dataset schemas and services with semantics (Bowers, 2004). “Kepler has adopted the OWL web ontology language [OWL] (more specifically, OWL-DL) as the primary language for capturing domain-specific terminologies used in semantic metadata” (Berkley, 2005).

Kepler uses semantics to suggest semantically relevant services and data. During workflow composition it checks whether two connected services or data sources are ‘semantically compatible’ based on their annotations. It is stated in (Berkley, 2005) that “given a workflow service on the Kepler canvas, a user can search for all ‘semantically compatible’ resources (either datasets or services) that can be connected to the input (or output) of the service.”

BioExtract has no provisions for semantics.

2.4 Introduction To Galaxy

Until now, this chapter has given us a background on contemporary scientific frameworks. Keeping these ideas in mind let us now discuss Galaxy. Galaxy is mainly developed at Penn State with contributors from Emory, Harvard and University of California San Diego. It is funded by NSF, Penn State and the Huck Institutes for Life Sciences. This framework provides access to a number of bioinformatic data sources. It also provides various analysis tools for bioinformatics. It lets users add their own tools to their local instance of Galaxy.

There is a workflow component to Galaxy which lets you save workflows and share them with colleagues. It is highly popular in the bioinformatics community due to its simple and easy to use interface. However, Galaxy has no support for Web services and neither does it have any semantic extensions. Considering its popularity it seemed like an ideal candidate to make additions to. In the next few chapters we will discuss the workflow component of Galaxy, the Web service extensions we have added to Galaxy and how they are compatible with existing functionality of Galaxy.

CHAPTER 3

WORKFLOWS IN GALAXY

Galaxy provides a number of tools for performing analyses on bioinformatics data. In addition to this it also has a workflow application which lets the user arrange different tools in a logical workflow, save this workflow, edit and execute it. There are two ways in which workflows are created in Galaxy. It will be discussed in a later chapter how the Web service extensions can be utilized in these workflows. This chapter discusses the workflow functionality of Galaxy in detail.

3.1 Workflows From Scratch

As mentioned in the beginning of this chapter, there are two kinds of workflows in Galaxy. The first kind which do not require any user input at runtime are termed as ‘from scratch’ (Galaxy Screencasts, 2009) workflows. These workflows can be directly created on the Galaxy workflow canvas and executed. To create a workflow from scratch the user would go to the Galaxy workflow canvas and select the tools to be arranged in the workflow. Figure 3.1 shows a representation of a sample workflow.

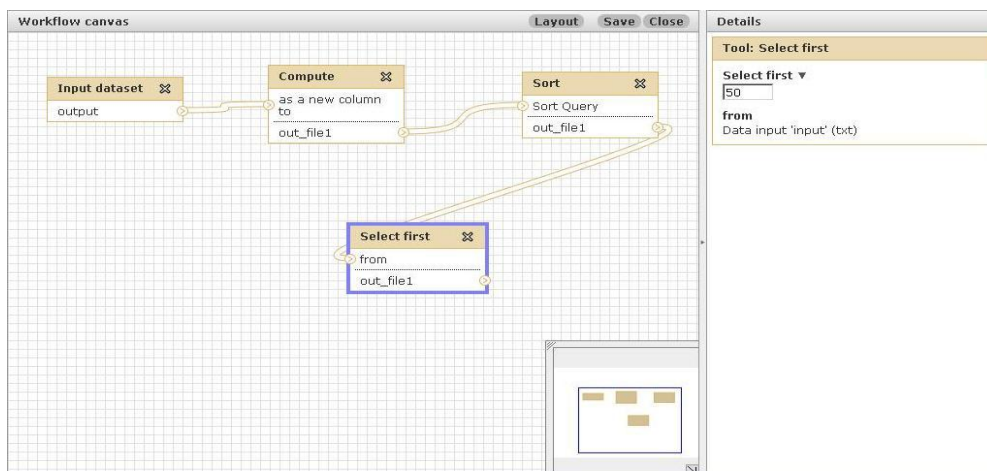


Figure 3.1 An example workflow in Galaxy

In this example, the input dataset is a 6 column Browser Extensible Data (BED) formatted file. A BED file has first 3 required columns, viz., chrom, chromStart, chromEnd. We will refer to the columns in a BED file as c1, c2, c3, etc. To find the length of an interval in a BED file, chromStart (c3) has to be subtracted from chromEnd (c1). The second tool in our workflow, ‘Compute’, computes the expression (c3-c2) for every row (interval) in the BED file to get the length of the interval and adds this value as c7 of the BED file. The third tool in our workflow sorts the BED file based on the values in c7 i.e., on the length of the intervals in descending order. The fourth tool selects the first 50 lines from this sorted BED file. Thus the result of this workflow is the 50 longest intervals for a given BED file. The inputs, to each of the tools in the Galaxy are known prior to execution. Thus, this workflow is saved with these inputs and can be executed directly.

3.2 Workflows From History

Workflow creation may not always be as simple as described in section 3.1. Many times the input/s of a tool can only be determined at run time. This brings us to the second type of workflow in Galaxy.

Galaxy lets you create workflows from history. Each tool that we run in Galaxy is saved in a local history that is visible to the user as the right panel in Galaxy. This means, the inputs as well as the results of each tool run are stored in Galaxy. Galaxy lets the user choose different tool runs from history to create a workflow.

Let us look at an example of such a workflow. In our example we query EuPathDB (Aurrecoechea, 2009; Aurrecoechea, 2006) to get all the genes for organism ‘Plasmodium falciparum’.

1	2	3	4
[Gene]	[Genomic Location]	[Product Description]	[Organism]
malmt0_rna_16: rRNA	H76611: 3 - 33 (+)	null	PLasmc
malmt0_rna_20: rRNA	H76611: 34 - 71 (+)	null	PLasmc
malmt0_rna_9: rRNA	H76611: 72 - 125 (+)	null	PLasmc
malmt0_rna_17: rRNA	H76611: 126 - 165 (+)	null	PLasmc
malmt0_rna_LSUC: rRNA	H76611: 204 - 226 (-)	null	PLasmc

Figure 3.2 Data from EuPathDB

As we see in Figure 3.2 the result has 5616 lines. The columns are Gene, Genomic Location, Product Description and Organism. Now if we want to look at only those genes which have a description. We choose the ‘Filter data on any column using simple expressions’ tool from the ‘Filter and Sort’ toolset in Galaxy.

Filter

Filter:
 ▼
 Query missing? See TIP below.

With following condition:

 Double equal signs, ==, must be used as shown above. To filter for an arbitrary string, use the Select tool.

Figure 3.3 Filtering only those genes which have a description

In Figure 3.3 we filter on c3, i.e., column 3, the Product Description column. Now if we wish to see only the Gene and Product Description columns, we can use the ‘Cut columns from a table’ tool from the ‘Text Manipulation’ toolset in Galaxy.

Cut

Cut columns:
c1,c3

Delimited by:
Tab

From:
2: Filter on data 1

Execute

Figure 3.4 Cutting columns c1 and c3 from the data

Figure 3.4 shows how we have chosen column c1 and c3 as the columns in our output.

History Options

refresh | collapse all

Unnamed history

3: Cut on data 2 5,405 lines, format: tabular, database: ?

Info:
save | rerun

1	2
[Gene]	[Product Description]
mal_mito_1	cytochrome c oxidase subunit 3
mal_mito_2	cytochrome c oxidase subunit I
mal_mito_3	apocytochrome b
PFA0005w	erythrocyte membrane protein 1, PfEMP1
PFA0010c	rifin

Figure 3.5 Output showing the Gene and description columns only

We can see in Figure 3.5 that our output has 5405 lines. Unless we know prior to looking at the data obtained in step1, that c3 refers to the Product Description column, we cannot enter the filter condition in the filter tool. Also, it could be possible that for some organism all the genes in the data have a Product Description. In such a case the Filter tool would be unnecessary. Hence, it would not be possible to do a ‘from scratch’ workflow for the above set of tools. However, we may still want these tools in a workflow so that we do not need to execute these three individual tools each time we want to perform these same operations. For this we create a workflow from the history in Galaxy. We do this by choosing the ‘Extract Workflow’ history option. This lets you choose the tools from history that you wish to combine in a workflow.

The following list contains each tool that was run to create the datasets in your current history. Please select those that you wish to include in the workflow.

Tools which cannot be run interactively and thus cannot be incorporated into a workflow will be shown in gray.

Workflow name

Tool	History items created
EuPathDB <i>This tool cannot be used in workflows</i>	1: EuPathDB <input checked="" type="checkbox"/> Treat as input dataset
Filter <input checked="" type="checkbox"/> Include "Filter" in workflow	2: Filter on data 1
Cut <input checked="" type="checkbox"/> Include "Cut" in workflow	3: Cut on data 2

Figure 3.6 Choose which tools you need in the workflow

Galaxy then creates a workflow using the tools you selected. You can view this workflow and edit it like any simple workflow as shown in Figure 3.7

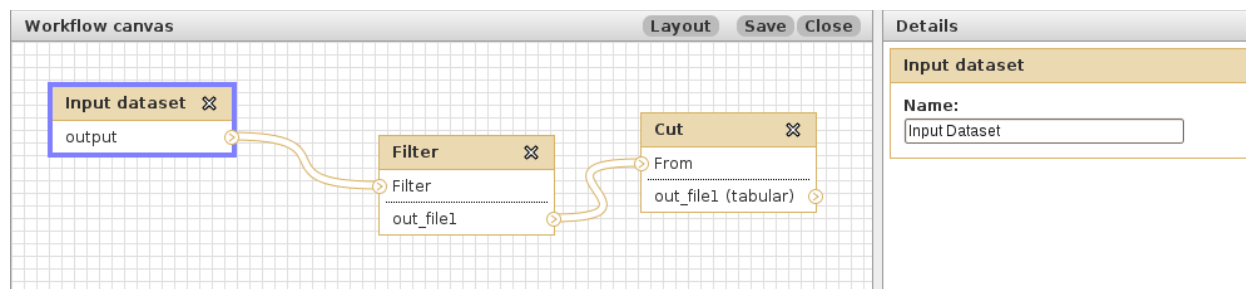


Figure 3.7 Workflow created from history

Before executing the workflow, the user can edit the input parameters of the tools if needed or the workflow can be executed as it is.

In this chapter, we have familiarized ourselves with the concept of tools and workflows in Galaxy. Tools are programs or scripts which fetch data from a remote or local resource or perform a certain computation on the given data. A workflow is formed when these tools are connected in a logical flow to reach a particular goal.

The next chapter will explore the Web service functionality we have added to Galaxy.

CHAPTER 4

INVOKING WEB SERVICES THROUGH GALAXY

As mentioned in Chapter 2, though Galaxy is popular framework for bioinformatics it lacks Web service functionality. There are two approaches through which this functionality can be added to Galaxy. The first approach deals with modifying the source code of Galaxy (Wang, 2009). The second explores the idea of adding these extensions as tools to Galaxy. For developing our extensions we have chosen the latter approach. In the next section, the pros and cons of both approaches are discussed.

4.1 Approaches To Integrating Web Service Functionality

Galaxy is an open source project and the source code is available on their Website. The most natural approach to adding functionality to Galaxy would be to make changes to the source code. Though time would have to be invested to study the source code the advantage of this approach is that the developer is in complete charge of how the extension behaves. While this seems like an intuitive approach, in reality Galaxy is ever evolving and so is the source code. By the time changes are made to one version of the source code, another version may be released. It might not always be simple to integrate the changes made in the previous version in to the next. Also, the code is structured such that in order to reflect changes in one module, changes have to be also made to a number of interdependent modules. Hence, the biggest disadvantage of this approach is maintainability across versions.

This brings us to the second approach. Galaxy lets users add their own tools to the framework. The tools can be command line tools written in any programming language. An XML file describing the tool has to be created and added to a Galaxy folder. The location of this XML tool file has to be added to `tool_conf.xml` file to Galaxy. Other than the XML files there does not need to be any interaction with Galaxy source code. This will definitely do away with the disadvantage of the first approach. The disadvantage to this approach is that the behavior of the extensions is restricted to the pre-defined behavior of tools in Galaxy. However, this restriction also guarantees that our extensions will work with other tools in Galaxy as well as the workflow module of Galaxy. Another advantage is that users are already familiar interacting with tools in Galaxy and will easily adapt to using the added extensions.

Thus, weighing the relative pros and cons of the two approaches, we have decided to go with the second approach, since it guarantees that our extensions will be compatible with different versions of Galaxy code without sacrificing too much of the functionality.

In the rest of the chapter, the Web service extensions will be explained in detail with examples.

4.2 Background On Web Services

W3C defines a Web service as “a software system designed to support interoperable machine-to-machine interaction over a network” (Booth, 2004). This software system is nothing but an application programming interface (API) that can be accessed remotely over the Internet. There are two kinds of Web service protocols, SOAP and REST. We discuss these briefly here.

Every SOAP Web service has a description document attached to it viz. Web Service Description Language (WSDL) document. A WSDL is an XML file which describes the Web service in terms of different operations it performs, the inputs expected for the operations, the

outputs provided by the operations, the data types of these inputs and outputs and the location of the service.

Analogous to WSDL for SOAP Web services, there exists a Web Application Description Language (WADL) for RESTful Web services. WSDL2.0 (cite WSDL2.0) can also describe REST Web services, however, neither WADL nor WSDL2.0 are widely used to describe REST Web services. This means that every RESTful Web service need not have a description associated with it. Hence, to know the details of a RESTful Web service (in absence of a description document) the Web service itself has to be studied to determine its operations and input and output parameters. For the purpose of our extensions to Galaxy, we assume that the user knows these details about the RESTful Web service, since it cannot be guaranteed that the needed RESTful Web service has a description document attached to it. However, for the SOAP Web service extension we access the corresponding WSDL to give the user all the details about the SOAP Web service.

4.3 Invoking SOAP Web Services

In order to invoke a SOAP Web service operation we need a SOAP client. A SOAP client is responsible for building a SOAP message containing the inputs to the Web service. This SOAP message is sent to the Web service server over the Internet using an application protocol like Hyper Text Transfer Protocol (HTTP). The Web service then processes the inputs, builds a SOAP message containing the output and sends it back to the client.

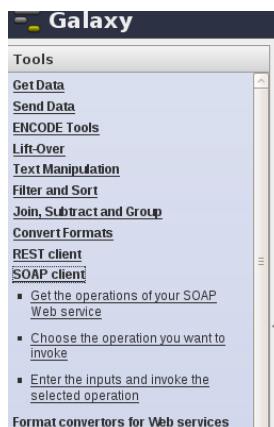


Figure 4.1 SOAP Web service client extension to Galaxy

We will now discuss the SOAP client extension added to Galaxy. In Figure 4.1 we see that the SOAP client appears in the Tools section of Galaxy. The SOAP client has 3 steps.

- Get the operations of your SOAP Web Service
- Choose the operation you want to invoke
- Enter the inputs and invoke the selected operation

In the first step the user inputs the (Universal Resource Locator) URL to the WSDL of the SOAP Web service. In our example, we are using the WSDbfetch Web service hosted by European Bioinformatics Institute (EBI) (European Bioinformatics Institute, 2009).

Get the operations of your SOAP Web service

Input the WSDL URL:

Input the URL of the WSDL. The output data file will contain the operations of the Web service

Figure 4.2 Enter the URL of the WSDL

In Figure 4.2 we see the SOAP client interface. On clicking the Execute button, the client fetches the WSDL and parses it to find a list of operations that the Web service performs. The output of this step is a tabular data file containing the list of operations of the Web service.

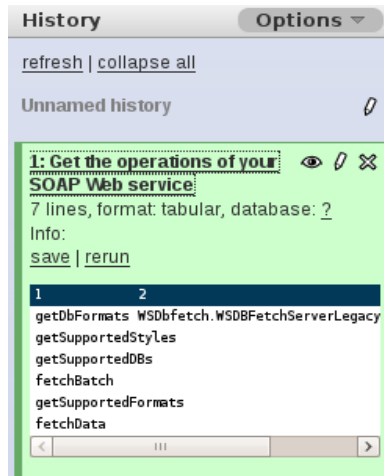


Figure 4.3 Output of the first step of the SOAP client

Figure 4.3 shows the output. Column 1 of the output file shows the operations. The client handles SOAP messaging through the Zolera SOAP Infrastructure (ZSI) libraries (Salz, 2005). The `widl2py` script in ZSI creates a client stub from the WSDL definition. For each Web service that we invoke through our SOAP client, we create a python package which contains this client stub. In Figure 4.3, column 2 in the output gives the name of this client stub in the form 'packagename.stubname'. In this example the package name is `WSDbfetch` and the stub name is `WSDBFetchServerLegacyService_client.py`

The second step of the SOAP client lets the user choose the operation that is to be invoked.

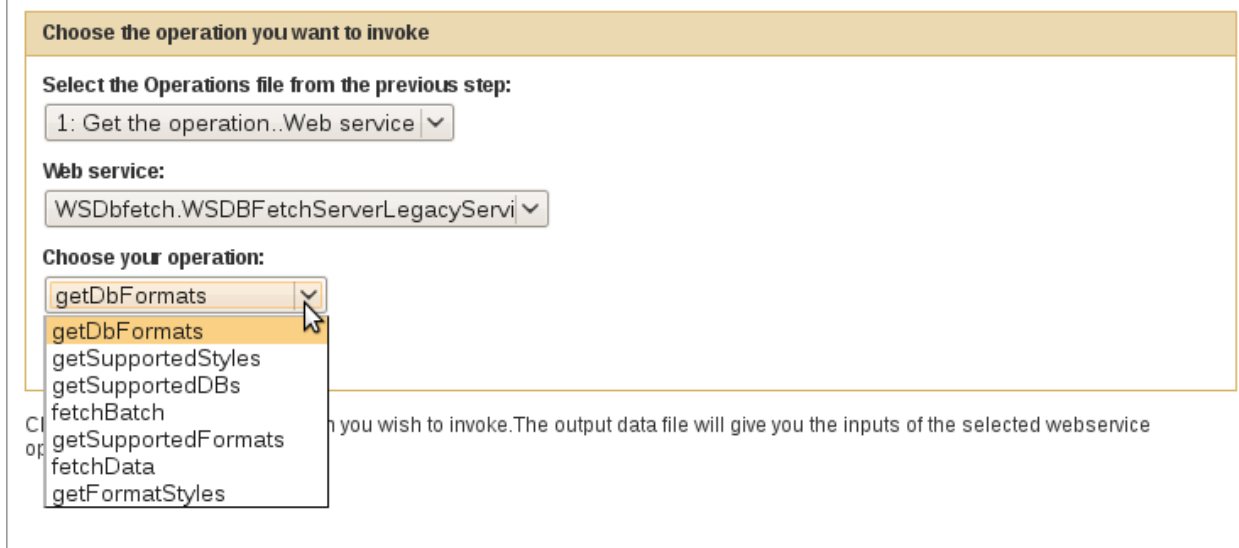


Figure 4.4 Operations of the Web service

In Figure 4.4 we see that user is shown all the operations of the Web service. The user can select any one operation by selecting it from the drop down menu. For the purpose of this example, let us select the fetchData operation. The output of this step is a tabular data file containing the inputs for this operation as well their data types.

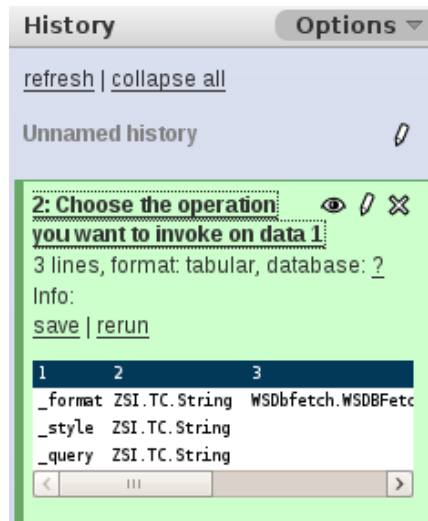


Figure 4.5 Output of the second step of the SOAP client.

We see in Figure 4.5 that column 1 of the output are the input parameters to the operation, column 2 are the data types for these inputs and column 3 is the name of the client stub in the form 'packagename.stubname' as described in Step 1 output.

In the third and final step of the SOAP client, the input values are given by the user and the Web service is invoked. Figure 4.6 shows how input values can be given to the Web service. The three input parameters format, query and style have been given input values of 'fasta', 'UNIPROT:ADH1A_HUMAN' and 'raw', respectively. The service API description on the EBI website explains what kind of input values are expected for the three inputs. In our example we wish to fetch an entry with id 'ADH1A_HUMAN' from the UNIPROT database. We want the entry to be in FASTA format and raw style.

Enter the inputs and invoke the selected operation

Select the Inputs file from the previous step:
2: Choose the operation on data 1

Web service:
WSDbfetch.WSDBFetchServerLegacyServi

Operation:
fetchData

Inputs

Inputs 1

Choose your inputs:
_format

Choose your datatype:
ZSI.TC.String

Do you need to upload data?:
No

Enter the input value:
fasta

Remove Inputs 1

Inputs 2

Choose your inputs:
_query

Choose your datatype:
ZSI.TC.String

Do you need to upload data?:
No

Enter the input value:
UNIPROT:ADH

Remove Inputs 2

Inputs 3

Choose your inputs:
_style

Choose your datatype:
ZSI.TC.String

Do you need to upload data?:
No

Enter the input value:
raw

Remove Inputs 3

Add new Inputs

Execute

Input the values of the inputs of the selected operation. Make sure that the values are of the correct datatype.

Figure 4.6 Invoking the Web service

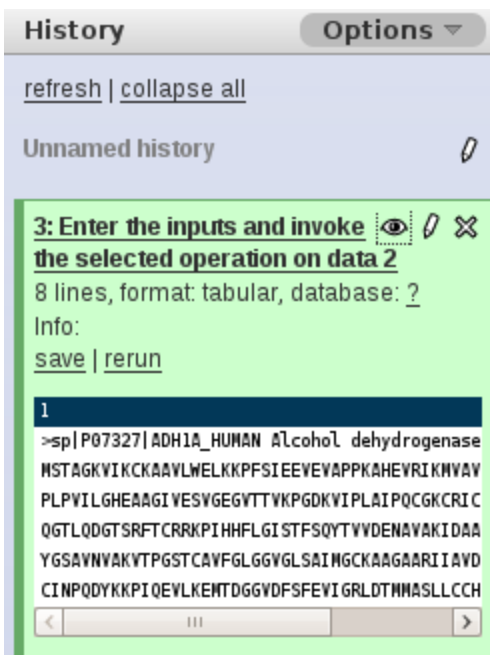


Figure 4.7 Output of the chosen operation

Figure 4.7 shows the output of our fetchData operation. The output data seems to be in a FASTA format, however, we also see a column number. This is because our SOAP client saves all output data in tabular format. At present Galaxy recognizes this output data as tabular data and not FASTA formatted data. Hence, to store this data as a FASTA file we will use the ‘Format convertors for Web services’ toolset.

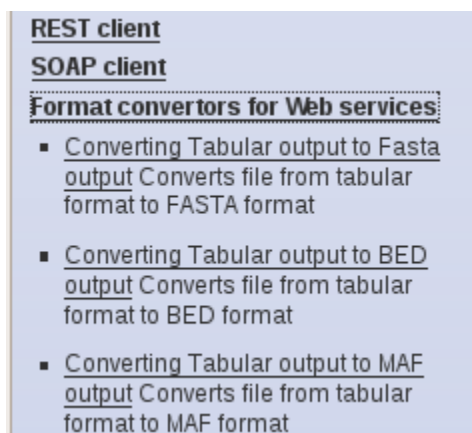


Figure 4.8 Format convertors for Web services toolset

This toolset saves the tabular output from the Web service into FAST Alignment (FASTA), BED or Multiple Alignment Format (MAF) outputs depending on the tool chosen. For our example, we will choose the ‘Converting Tabular output to FASTA’ tool.

The output of this tool is shown in Figure 4.9



Figure 4.9 File saved in proper FASTA format.

In the above example, we saw that the SOAP client is made up of three steps. The reason behind this design is that Galaxy does not allow a tool to read data during run time. Hence it is necessary to complete execution of one step, store its results and then access those results as input to the next step. Had the extensions been implemented using the first approach discussed in section 4.1, it would have been possible to add the SOAP client as a single step.

4.4 INVOKING RESTful WEB SERVICES

RESTful Web services are simpler to invoke than SOAP based Web services. Given the URL of the REST Web service GET, POST, PUT or DELETE HTTP requests are passed from the client to the REST Web service server. There are no operations besides these four defined for a REST Web service. However, as explained earlier in this chapter, REST Web services may not

have a description document and hence the RESTful Web service client does not have access to input parameter names. Generally, such information might be displayed for the user on the Website which gives the link of the RESTful Web service.

Let us now look at the RESTful Web service client added to Galaxy. The example we use for explaining the functionality of the REST client is similar to the example we used for the SOAP client. In this example, we fetch an entry from the European Molecular Biology Laboratories (EMBL) database in FASTA format and raw style.



Figure 4.10 RESTful Web service client extension to Galaxy

Figure 4.10 shows the REST client tool in Galaxy. We can see that it just has a single step unlike the SOAP client. Figure 4.11 shows the REST client interface in Galaxy. The Web service parameters as well as the values for the parameters are given by the user. Here the input parameters are db, id, format and style. The meaning of these parameters and the type of values they expect is described in the EBI webpage (<http://www.ebi.ac.uk/cgi-bin/dbfetch>). For our example, we choose 'embl' as our db value and 'x12399' as the id to an entry in the EMBL database. The format is FASTA and style is raw.

Universal REST Client

Enter the REST Webservice url:

Inputs

Inputs 1

Input the webservice parameter:

Do you need to upload data?:
 ▾

Enter the input value:

Inputs 2

Input the webservice parameter:

Do you need to upload data?:
 ▾

Enter the input value:

Inputs 3

Input the webservice parameter:

Do you need to upload data?:
 ▾

Enter the input value:

Inputs 4

Input the webservice parameter:

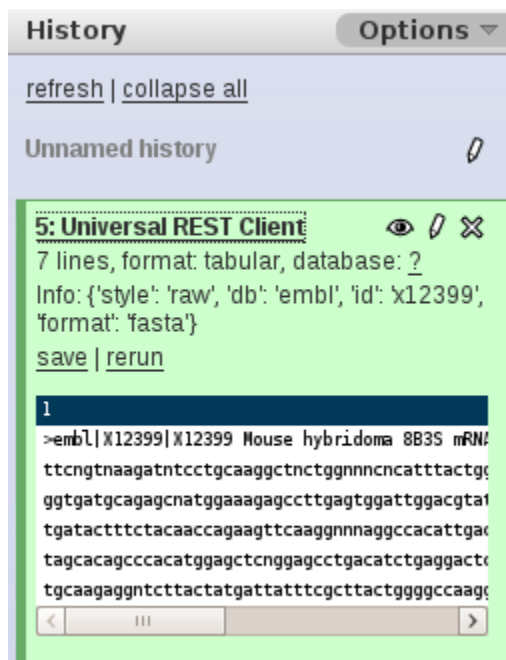
Do you need to upload data?:
 ▾

Enter the input value:

Input the URL of the webservice to be called followed by each parameter and value of the parameter you want to pass to the webservice

Figure 4.11 Inputs to RESTful Web service through Universal REST client

The REST client just like the SOAP client saves the output data in tabular format. Figure 4.12 shows the output for our example. This output will have to be saved as a FASTA file by using the Convertors for Web services toolset as shown in section 4.3.



The screenshot shows a 'History' window with an 'Options' dropdown. Below the window title, there are links for 'refresh' and 'collapse all'. The history is titled 'Unnamed history' with a trash icon. A single entry is visible, titled '5: Universal REST Client' with icons for view, edit, and delete. The entry details are: '7 lines, format: tabular, database: ?' and 'Info: {'style': 'raw', 'db': 'embl', 'id': 'X12399', 'format': 'fasta'}'. Below this, there are links for 'save' and 'rerun'. The main content area shows a table with one row, labeled '1', containing a FASTA sequence: '>embl|X12399|X12399 Mouse hybridoma 8B35 mRNA' followed by several lines of nucleotide sequence.

```

History Options
refresh | collapse all
Unnamed history
5: Universal REST Client
7 lines, format: tabular, database: ?
Info: {'style': 'raw', 'db': 'embl', 'id': 'X12399', 'format': 'fasta'}
save | rerun
1
>embl|X12399|X12399 Mouse hybridoma 8B35 mRNA
ttcngtnaagatnctctgcaaggctnctggnnncatttactgc
ggtgatgcagagcnatggaagagccttgagtggattggacgtat
tgatactttctacaaccagaagttcaaggnnaggccacattgac
tagcacagcccacatggagctcnggagcctgacatctgaggactc
tgcaagaggntcttactatgattatttcgcttactgggccaagc

```

Figure 4.12 Output of the Universal REST client

In this chapter, we have seen how the extensions can add Web service capabilities to Galaxy. In the next chapter we will discuss how these extensions can be enhanced by using semantics.

CHAPTER 5

UTILIZING SEMANTICS FOR GALAXY WORKFLOWS

In the previous chapter, it was demonstrated how we can presently add Web services to Galaxy. However, in this approach we have not used semantics of any kind. In this chapter, we discuss whether use of semantics can provide any significant advantages in workflow design as well as workflow execution for Galaxy.

5.1 Utilizing Semantics in Workflow Design.

“Scientific workflow tools like Kepler use domain ontologies for workflow design” (Berkley, 2005). This is needed in Kepler, since it is a multi domain workflow tool unlike Galaxy which is geared only towards the domain of Bioinformatics. However, even in the domain of bioinformatics, there exist multiple ontologies. While designing a workflow there are basically two kinds of issues that need to be considered. If the workflow consists of two tools T1 and T2, with T1 preceding T2 then, the output data type of T1 should match with the input data type of T2. The other issue is that, the output of T1 and the input of T2 should be the same conceptually. However, to determine whether the second requirement is satisfied, it is essential to semantically annotate the inputs and outputs of each tool or service in the workflow to assign meanings to them. For manual workflow design like Galaxy offers, the second requirement is not essential, since the user manually designs the workflow and it is assumed that the user has enough knowledge about the domain to conceptually match the inputs and outputs of Web

services. However, for Web service discovery or for semi-automatic workflow design it is essential that Web services or tools are semantically annotated.

Semantically annotated Web services can make adding Web services to workflows more convenient. SOAP Web services are described by WSDLs and our SOAP client is able to give some information about inputs and data type of inputs to the user. However, sometimes this information could be insufficient. For instance if there is an input parameter ‘format’ with data type ‘string’, it does not really tell the user much. Now if the user also knows that the ‘format’ is a property of the concept ‘Data File’ in an ontology, then he/she has a little more information about the values that could be assigned to the input. Semantic Annotations for WSDL (SAWSDL (Farrell, 2007)) can be used for this purpose.

For REST Web services, WADLs can give information regarding the method to be used (GET, POST, UPDATE or DELETE), input parameter names and data types. It can also give documentation about some sample input values. For REST Web services semantically annotated WADL (WADL-S) does not yet exist. However, if such a standard is developed there would be significant use of it in Web service workflow design. In the next section a preliminary model for WADL-S is proposed.

5.2 Semantic Annotations to WADL Documents

WADL documents are XML documents which describe RESTful Web services much like a WSDL document describing a SOAP Web service. Figure 5.1 shows an example of a WADL document (Hadley, 2009). While a WSDL describes various operations the SOAP Web service performs, a WADL document describes the resource which is being accessed. The operations in a REST Web service are fixed, GET, PUT, POST or DELETE. A REST Web service can have one or more of these operations. Before we discuss a model for semantically

annotated WADLs, it is at first necessary to familiarize ourselves with the tags contained in a WADL file. The <application> element is the root element.

```

1 <?xml version="1.0"?>
2 <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xsi:schemaLocation=http://wadl.dev.java.net/2009/02/wadl.xsd
4 xmlns:tns="urn:yahoo:yn"
5 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6 xmlns:yn="urn:yahoo:yn"
7 xmlns:ya="urn:yahoo:api"
8 xmlns="http://wadl.dev.java.net/2009/02">
9 <grammars>
10   <include
11     href="NewsSearchResponse.xsd"/>
12   <include
13     href="Error.xsd"/>
14 </grammars>
15
16 <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
17   <resource path="newsSearch">
18     <method name="GET" id="search">
19       <request>
20         <param name="appid" type="xsd:string"
21           style="query" required="true"/>
22         <param name="query" type="xsd:string"
23           style="query" required="true"/>
24         <param name="type" style="query" default="all">
25           <option value="all"/>
26           <option value="any"/>
27           <option value="phrase"/>
28         </param>
29         <param name="results" style="query" type="xsd:int" default="10"/>
30         <param name="start" style="query" type="xsd:int" default="1"/>
31         <param name="sort" style="query" default="rank">
32           <option value="rank"/>
33           <option value="date"/>
34         </param>
35         <param name="language" style="query" type="xsd:string"/>
36       </request>
37       <response status="200">
38         <representation mediaType="application/xml"
39           element="yn:ResultSet"/>
40       </response>
41       <response status="400">
42         <representation mediaType="application/xml"
43           element="ya:Error"/>
44       </response>
45     </method>
46   </resource>
47 </resources>
48
49 </application>

```

Figure 5.1 Sample WADL document.

The <application> tag contains the WADL description. The elements which form the main part of the WADL document are the <resources> and the <resource> elements. All WADL files contain these tags. A <resources> element can have one or more <resource> elements. The <resources> element also contains the base Uniform Resource Identifier (URI) of the resource. The <resource> element has a path attribute which is optional. When it is present it is relative to the base URI in resources and in combination with the base URI in resources provides a path to the resource. The <method> element is an optional element which describes the method by which the resource is to be accessed. The <method> element contains the <request> and <response> elements. <request> describes the input parameters for the method and <response> elements describe the <output> parameters for the method. A <response> is optional and may not always exist. Last the <param> element describes the input parameters that the method takes. Unless the 'required' attribute for a <param> element is true, the input parameter is optional. The <option> tag in parameter gives different possible input values that parameter.

Now that we have an understanding of the basic tags of WADL, we can move further to contemplate about semantic annotations to these tags. We have proposed the model for WADL-S based upon the initial idea of WSDL-S (Sivashanmugam, 2003; Rajasekaran, 2004). Table 5.1 shows the tags in WADL which can incorporate semantic annotations. Figure 5.2 shows a meta-model for WADL-S. The dotted rectangles indicate the WADL-S tags

Table 5.1 Proposed WADL-S semantic annotation attributes

WADL tag	Semantic attributes	Explanation
Resource	wadls:concept	Used to specify the meaning of this resource.
Method	wadls:concept	Used to specify what this operation does
Method	wadls:pre	Used to specify any preconditions that need to be satisfied for this operation
Method	wadls:post	Used to specify post conditions for the operation
Param	wadls:concept	Used to specify the meaning of this parameter

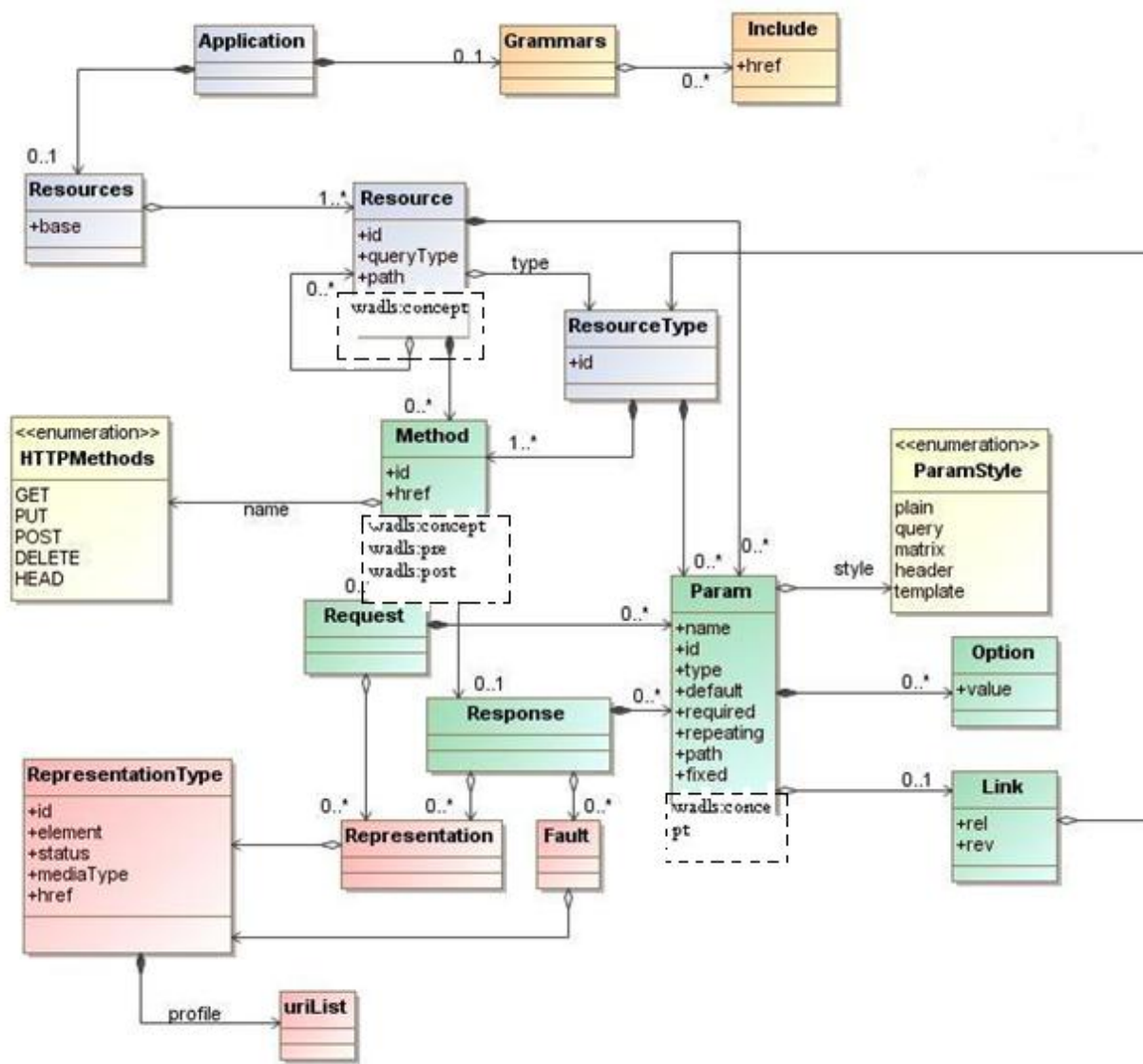


Figure 5.2 Meta-Model for WADL-S

5.3 Additional Semantics for Data and Process Mediation

In section 4.2 it was discussed how semantics could be utilized in workflow design and execution. In this thesis, the focus was on adding Web service capabilities to Galaxy with minimum impact on current Galaxy code. This approach has its advantages in that our extensions are independent of changes in the Galaxy code. However, another approach has been discussed

in (Wang, 2009). This approach discusses how semantics can help during Data and Process mediation.

This paper discusses a semi-automatic approach towards Web service workflow composition in Galaxy. The enhanced Galaxy workflow system discussed in this approach can provide Web service suggestions based on the current state and the overall goal of the workflow. These suggestions can either be backward suggestions, forward suggestions or even bi-directional suggestions and can be determined by a ranking algorithm. This process mediation assumes that Web services are described using SAWSDL/WSDL-S and it uses Web Ontology Language (OWL) and SWRL (Semantic Web rule Language) to specify the goal, pre-condition and effect for each component in the workflow.

For data mediation, their approach again uses semantics to align the input and output of Web services. If the input and output of Web services are simple type, there is relatively less chance of a mismatch. However, if they are complex types, some sort of data mediation is needed to map the messages between two Web service operations. This data mediation requires semantic annotation of the inputs and outputs of Web services which helps determine if they relate to the same concept in the ontology.

In this chapter, we have discussed different ways in which semantic annotations can benefit the workflow design, we have discussed an approach towards semantically annotating WADLs and we have looked at an existing approach where semantic annotations benefit data and process mediation.

CHAPTER 6

APPLICATION SCENARIOS

In this chapter, we will discuss two different scenarios in which we use the REST as well as SOAP extensions to Galaxy. We will use these extensions along with generic Galaxy tools and in a workflow and execute this workflow.

6.1 Workflow Using SOAP Client on EBI Web Service

In this scenario, we will be extending the example used in section 3.2. The SOAP Web service used in the example fetched data from the UNIPROT database based on an id. This data is a FASTA file. In this scenario, we will compute the sequence length of this FASTA output using the ‘Compute Sequence Length’ tool in the ‘FASTA manipulation toolset’ provided by Galaxy. The three steps of the SOAP client have to be composed in a workflow using Galaxy’s history feature. We will edit this workflow by adding two more tools, ‘Converting Tabular output to FASTA output’ and ‘Compute Sequence length’. Figure 6.1 shows our edited workflow.

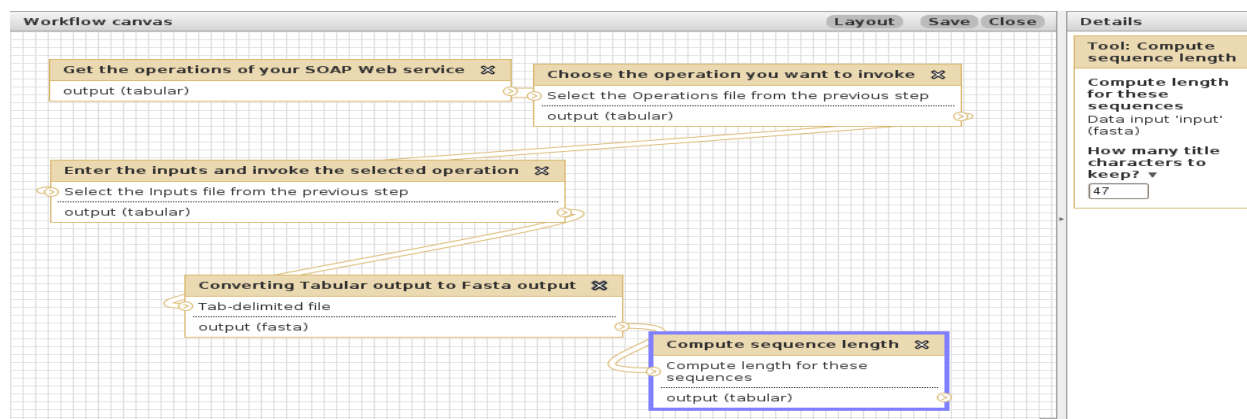


Figure 6.1 Edited workflow showing three steps of SOAP client and two additional tools

The ‘Compute sequence length’ tool requires input about number of title characters of the FASTA file to keep in the result. The user can change this number before running the workflow. In our example we choose 47.

```
sp|P07327|ADH1A_HUMAN Alcohol dehydrogenase 1A; 375
```

Figure 6.2 Output of the SOAP workflow

Figure 6.2 shows us the output of our edited workflow. We selected the 47 title characters and the sequence length computed by the tool is 375.

6.2 Workflow Using REST Client On EBI Web Service

In this scenario, we will be extending the example we used in section 4.3. We had extracted the entry for id x12399 from the EMBL database in FASTA format. In this scenario, we will use the REST client to obtain three FASTA sequences from the EMBL database with ids, ‘AE014292, AE017197 and AE017354’. After this we will use the Web service format convertor toolset to convert the tabular output into FASTA. Then we will create a workflow where we use the ‘Compute sequence length’ tool in the FASTA manipulation tool set of Galaxy to compute sequence length for the sequences of these three ids. The last tool in our workflow will be the ‘Filter by sequence length tool’, which we will use to filter any sequences that are

lesser than 1200000 characters in length. In Figure 6.3, we see the workflow for the above mentioned scenario. Figure 6.4 shows the output of steps 3 and 4 of the workflow. Step3 shows the sequence lengths for the three ids that we have chosen.

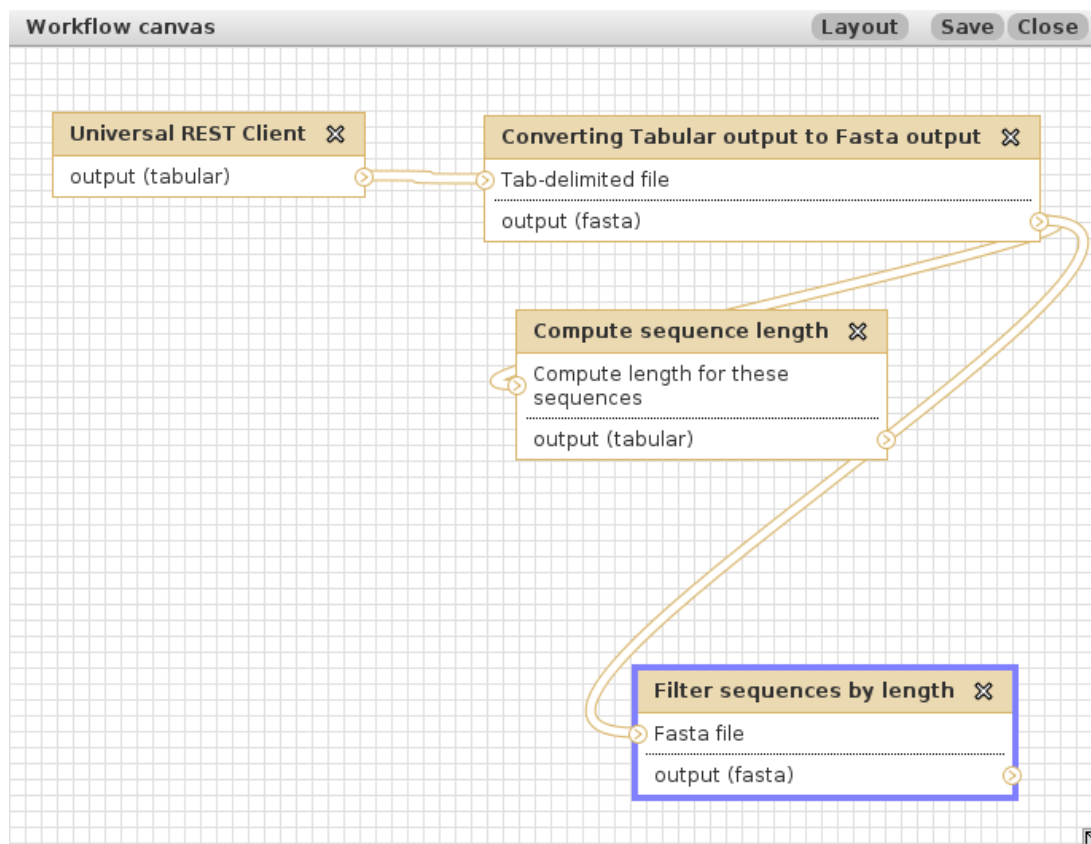


Figure 6.3 Workflow using REST client and FASTA manipulation tools

History Options

refresh | collapse all

Unnamed history 0

Add tags to history

4: Filter sequences by length on data 2 eye edit delete

4.5 Mb, format: fasta, database: ?

Info:

Save | rerun

```
>embL|AE014292|AE014292 Brucella suis 1330 chromosome II, complete
gcgcaaaagaaaaaggccccaaacgttgccgtcgtggaagcccttctcattggttagc
agctagagaatcgcaaatccacgaatcaactgtcaagtgtatttaacgtcaattggcggg
tggatttcttttgcccttggaaggtgaaagaaaaatggagattcaattggcctcgacgcc
gtttgacgtcgaaggatgtcgttcgccttattggcggcgagaaaaagccggcaaat
cccgaaggagctgctgccgacaagtggcagcttcatcgtgctatgcaagcgaaac
```

3: Compute sequence length on data 2 eye edit delete

3 lines, format: tabular, database: ?

Info:

Save | rerun

1	2
embL AE014292	1207381
embL AE017197	1111496
embL AE017354	3397754

Figure 6.4 Output of Step3 and Step4 of the workflow

In Step 4 we filter out all sequences which are lesser than 1200000 characters. Thus, in the output of the workflow we see only the sequences for AE014292 and AE017354.

In this chapter, we have seen that our extensions to Galaxy can interact seamlessly with native tools of Galaxy in a workflow.

CHAPTER 7

EVALUATION

For the purpose of testing the Web service extensions to Galaxy, we set up a local Galaxy instance and added the REST and SOAP clients as tools to this Galaxy instance. For testing the extensions we used various RESTful and SOAP Web services from the EBI web site. The reason for choosing EBI Web services was that it provided an extensive collection of both SOAP as well as REST Web services with good documentation.

We chose to add Web service extensions to Galaxy because Galaxy is a widely used Bioinformatics analysis framework and its many users can definitely benefit if Web service capability is added to it.

The reason for using SOAP Web services was obvious, since at present the number of SOAP Web services for bioinformatics is huge. Also, SOAP Web services have standard description which can help guide the user towards adding inputs for the Web service.

RESTful Web services have gained high popularity in recent years owing to their simplicity. Hence, it is expected that soon there would be increasing number of RESTful Web services for bioinformatics as well. Thus, we decided to include a REST client extension to Galaxy as well.

When compared to Taverna WSDL scavenger, we found that the interface of the Galaxy SOAP client is a lot more intuitive. Also, in the WSDL scavenger, there is no mention of the data type of the input value that a parameter takes. We provide this information in our SOAP client.

Adding external Web services to Taverna through the WSDL scavenger is also a three step process similar to the one we have for the SOAP client. However, for the SOAP client all the steps take place on the same window and the user does not have to juggle different windows or tabs. In Taverna the WSDL is entered on a pop up window, the WSDL then gets added to a list on the main Taverna window and finally when the user wishes to execute it, another window opens up where the user can edit the input values and execute the workflow. To see the result of the workflow, the user has to go back to the main Taverna window.

Adding a SOAP Web service to Galaxy workflow involves executing the three steps for the SOAP client and then using the 'Extract workflow' history option. On the other hand in Taverna, SOAP Web services can be added directly to a workflow.

There is nothing analogous to the REST client in Taverna. Taverna has no functionality for adding REST Web services to the workflow.

Evaluating our Web service extensions against similar functionality of Taverna, it appears that our extensions are easier to use and they also provide access to REST Web services, which is not available in Taverna.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this thesis, we have demonstrated how Web service capability can be added to Galaxy. It was the focus of this thesis to develop these extensions without affecting any native Galaxy code so that these extensions could be relatively independent to changes in the Galaxy code. Hence both these extensions are added as tools to Galaxy.

It has also been demonstrated that both RESTful and SOAP Web services can be invoked through Galaxy and once invoked, workflows can be created using these Web services and native Galaxy tools.

We introduced a preliminary model for semantic annotations to WADL or WADL-S. Though in this thesis we have only introduced the idea, it is hoped that such a standard is developed in the future.

In the future, the Web service extensions to Galaxy could be adapted to use semantically annotated WSDLs as well as WADLs to making it more convenient for the user to enter input values for the Web services. Also, some kind of caching mechanism could be developed which would cache the Web services that a user has already used, so the user need not enter the data for frequently used Web services.

REFERENCES

- Aurrecochea, C., J. Brestelli, B. Brunk, S. Fischer, B. Gajria, X. Gao, A. Gingle, G. Grant, O. Harb, M. Heiges, F. Innamorato, J. Iodice, J. Kissinger, E. Kraemer, W. Li, J. Miller, V. Nayak, C. Pennington, D. Pinney, D. Roos, C. Ross, G. Srinivasamoorthy, C. Stoeckert Jr., R. Thibodeau, C. Treatman and H. Wang (2009). EuPathDB: a portal to eukaryotic pathogen databases. *Nucleic Acids Research* , 38, Dzzz-Dzzz.
- Aurrecochea, C., M. Heiges, H. Wang, Z.Wang, S. Fischer, P.Rhodes, J. Miller, E. Kraemer, C. Stoeckert Jr., D. Roos and A. Gingle, G. Grant, O. Harb, F. Innamorato, J. Iodice, J. Kissinger (2006). ApiDB: Integrated Resources for the Apicomplexan Bioinformatics Resource Center. *Nucleic Acids Research* , 35, D427-D430.
- Berkley, C., S. Bowers, M. Jones, B. Ludascher, M. Schildhauer and J. Tao. (2005). Incorporating Semantics in Scientific Workflow Authoring. *The 17th International Conference on Scientific and Statistical Database Management*.
- BioExtract Server*. (2008). Retrieved from <http://bioextract.org/help/aboutWorkflows.html>
- Booth, D., H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard. (2004). Retrieved from Web Services Architecture: <http://www.w3.org/TR/ws-arch/>
- Bowers, S., K. Lin and B. Ludascher. (2004). On Integrating Scientific Resources through Semantic Registration. *International Conference on Scientific and Statistical Database Management*.
- European Bioinformatics Institute. (2009). *WSDbfetch Web service*. Retrieved from <http://www.ebi.ac.uk/Tools/webservices/services/dbfetch>
- Farrell, J. and H. Lausen (Ed.). (2007). *Semantic Annotations for WSDL and XML Schema*. Retrieved from <http://www.w3.org/TR/sawsdl/>
- Fisher, P. (2009). *Taverna Workflows*. Retrieved from <http://www.mygrid.org.uk/outreach/presentations/presentation-fisher2009a/>
- Galaxy Screencasts*. (2009). Retrieved from <http://galaxy.psu.edu/screencasts.html>
- Hadley, M. (2009). Retrieved from Web Application Description Language: <http://www.w3.org/Submission/wadl/>
- Lig H., C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao and H. Zheng (1999). Overview Of The Ptolemy Project. Dept. EECS, Univ. California Berkeley, CA, ERL Tech. Rep.UCB/ERL no.M99/37.
- 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.
- 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).
- McPhillips, T., S. Bowers, D. Zinn and B. Ludascher. (2009). Scientific workflow design for mere mortals. *Future Generation Comp. Syst.* , 25, 541-551.
- OASIS. (2009). Retrieved from OASIS: <http://www.oasis-open.org>

- 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.
- Rajasekaran, P. (2004). *Enhancing Web service Descriptions using WSDL-S*.
- Salz, R. (2005). *ZSI: The Zolera Soap Infrastructure*. Retrieved from <http://pywebsvcs.sourceforge.net/zsi.html>
- Sivashanmugam K., A. Sheth, J. Miller, K. Verma, R. Aggarwal and P. Rajasekaran. (2003). Metadata and Semantics for Web services and Processes. *Databases and Information Systems* , 245-271.
- Taylor, J., I. Schenck, D. Blankenberg and A. Nekrutenko. (2007). Using galaxy to perform large-scale interactive data analyses. *Current protocols in bioinformatics* , Chapter 10, Unit 10.5.
- Wang, R., D. Brewer, S. Shastri, S. Swayampakula, J. Miller, E. Kraemer and J. Kissinger. (2009). Adapting the Galaxy Bioinformatics Tool to Support Semantic Web Service Composition. *International Conference on Web Services*.

APPENDIX A

GALAXY WEB SERVICE EXTENSION USER'S GUIDE

This guide assumes that the user has installed the Galaxy Web Service extensions on their system. Download the WSExtensions.tar.gz from the <http://www.cs.uga.edu/~guttula>. After you unzip the folder, for installation instructions check the README.txt file in the installation folder.

This document is intended to be a step by step guide through the functionality of the Galaxy Web Service Extensions.

Locating the extensions

The REST client, SOAP client and Web Service Format Convertors will appear at the end of the Tools Menu.

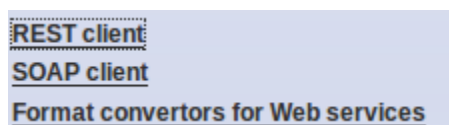


Figure 1 Web service extensions in Galaxy

The SOAP client

To execute a SOAP Web service through the SOAP client, we have to execute three steps. On clicking the SOAP client link in the Tools menu we see the three steps. These steps have to be executed **in order** for the client to function properly.

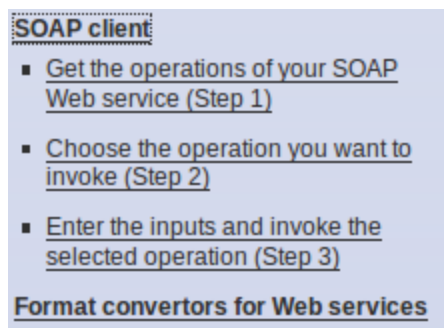


Figure 2 The 3 steps of SOAP client

Click on the first link and enter the URL for the WSDL file of the Web service. The result of this step will be a tabular file with the operations of the Web service listed.

Now let us move onto Step 2 by clicking the second link. From the ‘Choose your operation’ drop down list, choose the operation you want to invoke. The result of this step is a tabular file with the first column being the parameters for this operation and the second column being the data types of those parameters.

Note: If column 1 and column 2 of this file are blank, it means that this operation does not take any parameters.

Now click on the third and final step. In this step click the ‘Add new Input’ button for each parameter value you want to enter.

The ‘Inputs’ area has three drop down lists.

- From the ‘Choose your inputs’ drop down list select a parameter you wish to enter the value for.
- From the ‘Choose your datatype’ drop down list select the datatype corresponding to the parameter you have selected in the first drop down.
- If your input to the parameter is a data file in Galaxy history, select ‘Yes’ from the ‘Do you need to upload data?’ drop down list. This will give you a drop down list of all the data files in Galaxy history and you can choose the appropriate one.

- In case you do not wish to upload data, enter the input value for the parameter in the ‘Enter the input value’ text box.

After you have finished the above procedure for all the required parameters for the operation, click on execute. The result of this step is the output of the Web service stored as a tabular file.

The REST client

If you wish to invoke a REST Web service then click on the ‘REST client’ link in the Tools menu. Now click on the ‘Universal REST Client’ to see the interface.

Enter the URL for the REST Web service.

Click on the ‘Add new Inputs’ button to enter a Web service parameter name and value. For each parameter required by the Web service you will have to click the ‘Add new Inputs’ button. In the ‘Inputs’ area, do the following:

- In the ‘Input the webservice parameter’ textbox, enter the name of the parameter
- If your input to the parameter is a data file in Galaxy history, select ‘Yes’ from the ‘Do you need to upload data?’ drop down list. This will give you a drop down list of all the data files in Galaxy history and you can choose the appropriate one.
- In case you do not wish to upload data, enter the input value for the parameter in the ‘Enter the input value’ text box.

After you have finished entering values for all the required parameters, click the execute button. The result will be the output of the Web service stored as a tabular file.

Format convertors for Web services

The output of the SOAP client and REST client are tabular files. Sometimes the output of a Web service might be in a format other than tabular. In such cases you will observe that though the content of the output file is formatted in a different format (e.g., FASTA), Galaxy recognizes it

as tabular. In such a case you can convert this result from the Web service into either FASTA format by using the ‘Format convertors for Web services’ tools.

To do this, click on the appropriate convertor, choose the output file from the Web service stored in history and click on execute. This will save your file in the correct format. Same applies to BED and MAF formats.

Note: These convertors are different from the ‘Convert Formats’ tool supplied by Galaxy. You should use this tool only if the content of the file is in the desired format, but only saved as tabular.

Workflows using SOAP client

If you wish to include a SOAP Web service in a workflow, you will have to include the SOAP client in the workflow.

To do this we use the ‘Extract Workflow’ option from the History options of Galaxy.

First, we have to execute the three steps of the SOAP client individually. Then using the ‘Extract Workflow’ option we select those three steps while creating a workflow.

Once this workflow is created and saved, we can add any other tools in this workflow.

If the values to the SOAP Web service parameters need to be changed, you can click on Step 3 of the SOAP client and change the values.

Note: Once the SOAP workflow is created, values to its parameters can be changed.

However, a different operation than the one used while executing the client cannot be chosen.

Workflows using REST client

The REST client can be added to a Galaxy workflow like any other tool. No special considerations are required.

Thanks for using the Web service extensions tool. We hope you enjoy working with it!

APPENDIX B

GALAXY WEB SERVICE EXTENSION DEVELOPER'S GUIDE

This guide is intended for developers who wish to further enhance the Web service extension to Galaxy. It has technical documentation about the SOAP client, the REST client and the Web Service format convertors. It is assumed that the developer has set up an instance of the Galaxy server and the Galaxy Web service extensions.

The Web service extensions to Galaxy are all added as tools to Galaxy. Hence, their source code can be found in the tools folder of Galaxy. Also, each tool has an eXtended Markup Language (XML) file attached to it. To learn more about adding tools to Galaxy refer to:

<http://bitbucket.org/galaxy/galaxy-central/wiki/AddToolTutorial>

To learn more about various tags used in Galaxy tool xml files refer to:

<http://bitbucket.org/galaxy/galaxy-central/wiki/ToolConfigSyntax>

The SOAP client

Code Location: GALAXY_HOME/tools/SOAPclient

Modules: getOps.py, getInputs.py, invoke.py

Tool XML files: getInputs.xml, getOps.xml, invoke.xml

Packages used: clientGenerator

getOps.py

This module is the first step in the SOAP client.

This tool takes the WSDL URL as input from the user and writes a local file containing the operations of the Web service.

This module uses the creatorEngineComplex and wsdl2path modules from the clientGenerator package.

The wsdlUrl2path method from the wsdlLoader class in the wsdl2path module is used to execute the wsdl2py program that generates the client stubs for the WSDL in a given folder.

Thus, for every Web service the SOAP client invokes a client folder with the name of its WSDL is created in GALAXY_HOME/webservices folder. This client folder contains the client stubs created by wsdl2py.

The path2Ops method from the ClientCreator class in the creatorEngineComplex module is used to find the operations in the client stub of the Web service. This method returns dictionary containing the operation names of the Web service and the SOAP objects relating to the operation.

The resulting data file where the operations are written is a tabular file with the following details

Column 1: List of operations.

Column 2: Name of the client folder.

It is necessary to write the name of the client folder since, the input to our next step is this file and we need this information in the next step.

getInputs.py

This module is the second step in the SOAP client.

It takes the Web service client stub name, name of chosen operation and name output file as arguments.

This module uses the `creatorEngineComplex` module from the `clientGenerator` package.

It uses the `opname2inputs` method of the `ClientCreator` class in the `creatorEngineComplex` module which returns a dictionary of form `{inputname: typecode}` where `typecode` is the data type of `inputname`.

The resulting data file where the operations are written is a tabular file with the following details

Column 1: List of input parameter names.

Column 2: List of parameter datatypes

Column 3: Name of the client folder

Column 4: Name of the chosen operation

It is necessary to write the name of the client folder and the chosen operation since, the input to our next step is this file and we need this information in the next step.

invoke.py

This is the third and final step in the SOAP client.

This module takes the following as input:

- Name of the Web service client folder
- Name of the operation invoked

- Input parameter names
- Input parameter datatypes
- Input parameter values

The datatypes are only used to verify that the user has entered a valid value. A dictionary of the type {inputname:inputvalue}. This dictionary is passed to the invokeOp method.

The invokeOp method belongs to the ClientCreator class in the creatorEngineComplex module.

It invokes the operation and returns the Web service result.

This result is stored in a tabular file.

The REST client

Code Location: GALAXY_HOME/tools/restclient

Modules: univClient.py

Tool XML files: RESTclient.xml

univClient1.py

This module accepts the following input from the user:

- URL to the Web service
- Parameter name
- Parameter value

A dictionary of the type {paramname:paramvalue} is created. This dictionary is then URL encoded and passed to the Web service through the urllib.urlopen method.

The results of the invocation are written to a tabular file.

Also, this client maintains a history XML file (createdServices.xml) which contains a list of REST Web services that have been invoked by the client with their parameter names.

Web Service Format Convertors

Code Location: GALAXY_HOME/tools/Webservice_format_convertors

Modules: formatConvertor.py

Tool XML files: tabTofasta.xml, tabTobed.xml, tabTomaf.xml

This module takes two inputs, the file to read from and file to write to.

In the tool xml files for these tools, it is mentioned that the input type is tabular and the output type is either fasta, bed or maf.

This module only saves a tabular file with .fasta, .bed or .maf extensions.