Adapting the Galaxy Bioinformatics Tool to Support Semantic Web Service Composition

Rui Wang, Douglas Brewer, Shefali Shastri, Srikalyan Swayampakula, John A. Miller, Eileen T. Kraemer and Jessica C. Kissinger
University of Georgia, Athens, GA, 30602

{wang@cs., brewer@cs., shastri@cs., srikal@cs., jam@cs., eileen@cs., jkissing@ }uga.edu

Abstract

As the availability of Web services for the biological domain increases, the need emerges for a Web service composition designer that is easy for biologists to use. Our work focuses on providing biologists and bioinformaticians with an online, semantic Web service composition tool. We adapt a bioinformatics tool called Galaxy, to support semantic Web service composition. A semi-automatic approach for semantic Web service composition is utilized. An easy to use online interface is provided.

1. Introduction

Increasingly, Web services for applications in biological domains are available from resources such as the National Center for Biotechnology Information (NCBI), the European Bioinformatics Institute (EMBL-EBI), the DNA Data Bank of Japan (DDBJ) and the Protein Data Bank of Japan (PDBJ). Sophisticated users often wish to make use of several of these services in conjunction. Web service composition enables the use of multiple Web services in combination with one another. For biologists and non-computer scientists, it can be very challenging to create Web service compositions with the currently available designers.

At present, Web service composition designers are categorized either as mashup editors or as Business Process Execution Language (BPEL) editors. A mashup is a Web application that combines data from more than one source into a single, integrated tool, and the use of the term mashup typically implies easy, fast integration through open APIs and data sources, to produce results that are beyond the original goal of the data owners. Mashup editors (e.g., Yahoo Pipes, Microsoft Popfly, Google Mashup Editor) provide a visual interface through which the user may drag and drop components into a Web application.

BPEL editors (e.g., ActiveBPEL, NetBeans BPEL, Oracle BPEL, and Eclipse BPEL) are used to design traditional Simple Object Access Protocol (SOAP) based Web services; while mashup editors are used for lighter weight services typically Representation State Transfer (REST) based Web services. All of these designers provide intuitive graphical user interfaces. However, in keeping with the theme of simplicity associated with RESTful services, the mashup editors tend to be easier to use. In addition, they also utilize Web 2.0 technologies to make them available on the Web.

Although mashup editors are easier to use, they have limited scope for control flow and automation compared to BPEL editors. Our goal is to build a designer that supports both SOAP and REST based services, which includes the best features from both worlds. In other words, we would like to keep the simplicity of the REST based approach, while adding more options for data mediation as well as greater opportunities for automating the design of the process.

Our approach is to use semi-automatic composition. Semantics are introduced to reduce the burdens of Web service composition on the human designer by increasing the automation level of the Web service composition process.

In trying to find a suitable Web-based interface we were fortunate to have a colleague in Genetics who was familiar with Galaxy [1]. Galaxy is an integration tool in the bioinformatics domain that provides a friendly Web based interface similar to Yahoo Pipes.

1.1 Motivating Example

Since European Bioinformatics Institute (EBI) Web services are popular in the biological domain, we picked two of them for our evaluation, namely: WSDbfetch and WSWUBlast. The second one has a more complex type in the XML schema compared to the first one. The URLs of their WSDLs are as below:

http://www.ebi.ac.uk/Tools/webservices/wsdl/WSDbfetch.wsdl

http://www.ebi.ac.uk/Tools/webservices/wsdl/WSWUBlast.wsdl

Bioinformatics has many domain specific data formats, e.g., FASTA, GenBank, EMBL, BED, GFF, MAF, etc. Unfortunately, this forces biologists and bioinformaticians to do extra work to convert data

formats to fit different software. Therefore, the motivating sample workflow we choose is to convert the FASTA format output of the Web service WSDbfetch to the Galaxy tabular format. The fetchData operation of the WSDbfetch Web service retrieves data from the database specified in user inputs. The outputs of fetchData operation are fed into the Galaxy tool called "Fasta-to-Tabular" which can be found under the "Convert Format" category on the left panel of the user interface. This tool will convert the FASTA format data into tabular format data.

FASTA is a text-based format for representing DNA/Protein sequences. A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol at the beginning.

The remainder of this paper is organized as follows: Section 2 presents related work on Web service composition as well as a quick review of Galaxy. Section 3 shows how we adapt Galaxy to support Web services. Section 4 covers our extensions to Galaxy's current workflow features to support Web service composition. Section 5 describes our approach for adding semantics to Web service descriptions to facilitate process and data mediation. This is followed in section 6 that gives an overview of our implementation. Section 7 presents a preliminary evaluation of our composition design tool. Finally, section 8 discusses our conclusions as well as directions for future work.

2. Background and related work

2.1. Use of the Galaxy integration tool

Galaxy [1] is an open source framework for integrating data and tools in the biological domain. Through Galaxy, biologists can access popular data sources and a variety of useful data analysis tools. Figure 1 shows the interface hosted by the original Galaxy Web site. The left panel contains hyperlinks to data sources and tools. When a user clicks on a link, the associated content is displayed in the middle panel. The right panel displays the user's history.

WS-BioZard [2] was our previous attempt to develop a semantic Web service composition framework for the bioinformatics domain. It was built on earlier work in the METEOR-S project [3，4], which focused on semantic Web services. WS-BioZard supports semi-automatic Web service composition for biologists. Though WS-BioZard had many of the features desired, its user interface was not Web-based and not as intuitive as our current approach.

Galaxy lacks Web service capabilities as well as an effective way to utilize semantics. Since Galaxy is a larger and more mature project, we chose to add functionality to Galaxy. By using Galaxy, we will also benefit from the many data sources and types that are already integrated, tested and exercised in its code base.

Galaxy has an extensible framework for adding tools. However, adding new tools to Galaxy requires a local Galaxy server. As Galaxy is an online application, requiring the installation of a local Galaxy server is not desirable. Thus, in addition to tools, Web services would be a good solution for making Galaxy more extensible. A user wishing to use a Web service need not install a Galaxy server, but only provide the URL of the Web Service Definition Language (WSDL) file.

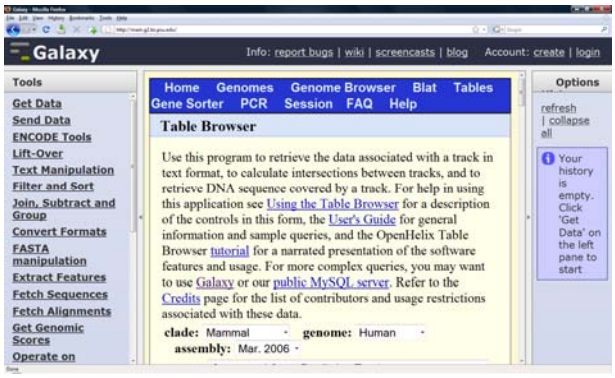


Figure 1. Galaxy interface

2.2. Related work

Web service composition approaches can be categorized into three categories [5]: manual, semi-automatic and automatic approaches.

Most popular Web service composition designers, including BPEL designers and the Taverna workflow designer (http://taverna.sourceforge.net/), employ a manual approach. That is, users manually select services and compose them together into a workflow. Taverna is the only designer targeted specifically for the biological domain. It is based on the Simple Conceptual Unified Flow Language (Scufl) rather than BPEL. However, since manual approaches have a longer learning curve than is desirable for many users, our approach, with the help of semantics, provides greater assistance to users composing Web services. Our goal is to substantially lower the learning curve for users and to also decrease the time it takes to develop a composition.

Automatic approaches, in which techniques from Artificial Intelligence (AI) such as classical and decision-theoretic planners that are used to automatically generate a composition of Web services, have been studied in academia for some time [6-9]. However, many researchers, e.g., Charif-Djebbar et al. [10], Hull et al. [11], Rao et al. [12] claim that these approaches are currently not mature enough and have too many limitations to be used in the real world, e.g., an expert is needed to encode initial state and goal state formally for

each workflow as well as annotate Web services with precise preconditions and effects. Analogous to our approach, Thakkar et al. [7] developed a framework for integration targeted at the biological domain. However, compared to our approach, it does not support operations on heterogeneous data, and so thus cannot deal with complex XML structures, while the data mediator in our system can handle complex XML structures.

For our purposes, a semi-automatic approach, which introduces semantics during the Web service composition process and also allows the user to interactively control the generated workflow, is in our view more practical than a manual or an automatic approach. Approaches described by Xu et al. [13] and Michael et al. [14] also fall under the semi-automatic service composition category. Especially relevant is [14], since it targets the biological domain. However, it focuses on applying semantics only to process mediation. Specifically, semantics are used to select a more suitable Web service during the assembly process. It does not discuss handling heterogeneous data mapping between Web services. This can be handled by our designer. Also in our approach we apply semantics to data mediation as well as process mediation.

3. Adding Web services to Galaxy

Our first extension to Galaxy adds the capability to invoke Web services. This is already completed and the evaluation is shown in the evaluation section. We do this by automatically generating a Web service client and then wrapping it in a small program so that Galaxy can invoke it as a tool. A Web service client is a program used to communicate with a specific Web service. The Zolera Soap Infrastructure (ZSI) (a Python package) is used here to help generate the Web service client.

Figure 2 shows the structure of WS Adapter that supports the integration of Web services. To add a Web service, users only have to provide the WSDL file, which describes the interface of the Web service. The WSDL2PY module will parse the given WSDL file and generate the client stubs. These stubs are a set of files containing the code required to interact with the Web service. Using the client stubs, the Python Introspection module will identify all the operations, data types, etc., thus helping the WS Client Factory generate the client. A Web service generally has multiple operations. When a user wishes to invoke any operation of an added Web service, the WS Invocation Handler module will create an instance of the service interface. In the meantime, the Input/Output XML Handler will parse the input data types and provide them to the user interface. After the user enters the input data, the WS Invocation Handler will feed the input data to the service instance that was created before and then invoke the operation of the Web service through the Internet. The returned SOAP message will be deserialized into Python objects which represent the data structure of the response. The Input/Output XML Handler will then parse these objects, so that they can be displayed on the user interface.

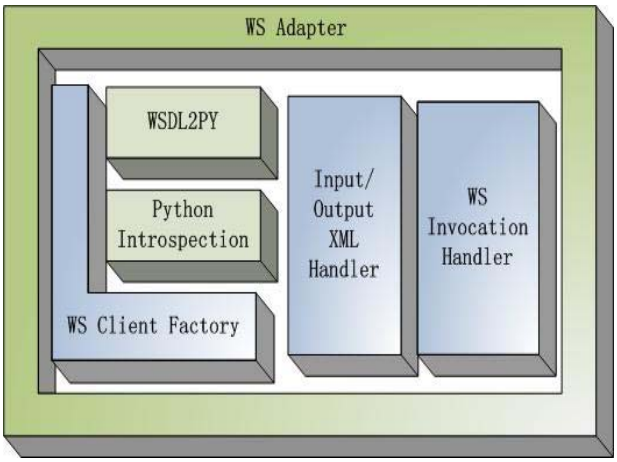


Figure 2. Architecture of WS Adapter (automatically add Web services to Galaxy)

4. Workflow capabilities of Galaxy

4.1. Current release

The latest release (Rev 1733:a4214de3752e) of Galaxy provides an online workflow editor. Through this editor, users can integrate tools preloaded in Galaxy into a workflow that is saved in the history and can be invoked as a normal tool in Galaxy. The input data stored inside the Galaxy server is either uploaded by the user (in the form of files) or retrieved from a database through Galaxy. This online editor interface is one of the main reasons that we chose to use Galaxy. Thus, we would like to keep this feature and extend it to support semantic Web service composition.

4.2. Adding Web Service Composition

With Web services integrated as tools in Galaxy, users have the ability to create workflows based on these Web services. Currently, we have already completed the functionality to support composition using both Web services and native tools of Galaxy. The test of this function is shown in the evaluation section. As a tool in Galaxy, a Web service can be manually composed with other tools in Galaxy to form a workflow. The resulting workflow can be saved to a user's history in Galaxy. The invocation of Web services and the execution of tools are considered steps in the user's history.

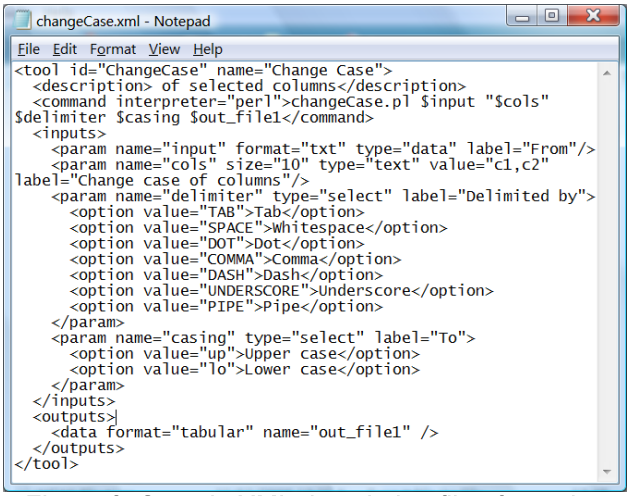
In addition, Web services with simple inputs/outputs can be composed into a workflow in the same way as Galaxy tools can be composed into a workflow. After a Web service is invoked, the Input/Output XML Handler will parse the returned SOAP message and deserialize it into Python objects. The simple type output is one string

or integer, etc. rather than a complicated data structure. Therefore, it is possible to feed this simple string into another Web service that takes a simple string as an input.

However, many Web services have complex types for their inputs and outputs. Thus, it is problematic to create a workflow involving such Web services. Semantics can be used for facilitating semi-automatic composition of the workflows by suggesting or plugging in appropriate Web services as well as handling data mediation for the user (see section 5).

We extended the workflow module of Galaxy to support XML. For this purpose we added a workflow composer module. The workflow composer invokes our extended workflow engine, which saves the workflow as an XML file locally on his/her machine or into his/her account. The workflow engine also executes this saved workflow when requested by the user. We are creating a Domain Specific Language (DSL) in Python called BpelPy for a useful subset of BPEL. It allows one to write executable process specifications as well as read and write BPEL.

Two types of workflows exist. The first kind of workflow is made up of only Web services. We save this type of workflow in a subset of standard BPEL format [15]. The resultant workflows from our Web service composition tool can be shared with a variety of BPEL engines or BPEL designers available today. The second type of workflow contains both Web services and native tools of Galaxy. In this case, we map the tool description to a SAWSDL like specification and then use a special invocation feature provided by BpelPy to execute native Galaxy tools. Every such tool native to Galaxy has its own XML description file. Figure 3 shows a sample of such a description file. This tool description file contains information about the input, the output, the command to invoke the program of the tool, etc.


changeCase.xml - Notepad

File Edit Format View Help

```
<tool id="ChangeCase" name="Change Case">
  <description> of selected columns</description>
  <command interpreter="perl">changeCase.pl $input "$cols"
$delimiter $casing $out_file1</command>
  <inputs>
    <param name="input" format="txt" type="data" label="From"/>
    <param name="cols" size="10" type="text" value="c1,c2"
label="Change case of columns"/>
    <param name="delimiter" type="select" label="Delimited by">
      <option value="TAB">Tab</option>
      <option value="SPACE">Whitespace</option>
      <option value="DOT">Dot</option>
      <option value="COMMA">Comma</option>
      <option value="DASH">Dash</option>
      <option value="UNDERSCORE">Underscore</option>
      <option value="PIPE">Pipe</option>
    </param>
    <param name="casing" type="select" label="To">
      <option value="up">Upper case</option>
      <option value="lo">Lower case</option>
    </param>
  </inputs>
  <outputs>
    <data format="tabular" name="out_file1" />
  </outputs>
</tool>
```

Figure 3. Sample XML description file of a native Galaxy tool

The various parts of an XML tool description file can be transformed to generate different parts of a WSDL file. The <tool> tag provides the tool id, name and version which can be included in the <definitions> tag of WSDL. The XML tool file takes inputs and produces outputs in Galaxy defined formats. We can express these Galaxy defined formats using different constructs in the XML Schema Definition (XSD) schema.

Once we define the input and output elements, we add the <wsdl:message> tags, <wsdl:PortType>, <wsdl:operation>, etc. and other WSDL tags to generate the WSDL file. The WSDL file then is semantically annotated using a Semantic Annotations for WSDL and XML Schema (SAWSDL) annotation tool like Radiant (http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1) to generate a SAWSDL file. These SAWSDL files are then used in normal BPEL compositions. For example, in Figure 3, the name of the first input parameter is "input". In our workflow, as shown below, it will be used in the <variable> tag to define a variable and then in the <copy> statement to copy values between two variables. To support this workflow model, we must extend the current workflow module as described earlier.

```
<variable name="inputVar" messageType=" inputs"/>
<variable name="fetchDataOutput" messageType="
fetchDataResponse"/>
……
<assign name="assign1">
  <copy>
      <from variable="fetchDataOutput"
                part="fetchDataReturn" />
      <to variable="inputVar" part="input"/>
  </copy>
</assign>
```

5. Utilizing semantic Web services

5.1. Semantic Web service

SAWSDL [16] provides mechanisms to annotate WSDL with semantic concepts, which can help disambiguate the description of Web services during automatic discovery or composition of Web services.

SAWSDL provides three types of semantic annotation: Model References, Lifting Schema Mappings and Lowering Schema Mappings. A Model Reference is used to semantically annotate different parts of the WSDL, such as the operation, fault, or the schema's complexTypes and simpleTypes to relate them to the ontology. Lifting and lowering schema mappings in SAWSDL apply only to schema definition languages and are used to facilitate automatic data mediation, so that two Web services can communicate (i.e., the output of one Web service can be transformed to the desired input

type/format of the second Web service). As long as the output of one Web service is semantically related to the input of another Web service, these schema mappings can be used to facilitate communication via data mediator [17].

5.2. Process mediation

Galaxy comes with many tools and users can also add more Web services to our extended Galaxy, so the number of services in Galaxy could grow quite high. Therefore, it is difficult for users to select appropriate services from the space and connect them in a suitable order to compose a workflow. This problem can be tackled by process mediation. Currently, this module is still under development.

In the current version of Galaxy and in other popular workflow designers (e.g., Taverna, NetBeans BPEL Designer, ActiveBPEL designer, etc.), their approaches to composition is essentially manual. We are following a semi-automatic approach based on the use of process mediators. Users can interact with the editor during service composition and can also ask for services to be suggested, at which time a set of ranked services will be provided to the user.

Our algorithm can provide forward, backward or bidirectional suggestions to users whilst they are using our editor to compose services into a workflow. A forward suggestion means that when a user picks the first service, the output of it can be fed into the suggested service. Conversely, a backward suggestion means that the output of the suggested service can be fed into the service already selected by the user. A bidirectional suggestion tries to suggest a service which can be plugged between two currently selected services.

In the case of a bidirectional suggestion, our ranking algorithm considers several criteria. It first looks at the compatibility of the inputs and outputs, i.e., the output of the prior service must match to the input of the suggested service, while the output of the suggested service must match to the input of the subsequent service. Then the algorithm will check the compatibility of preconditions and post-conditions/effects if available, i.e., the post-conditions from the prior service should imply the preconditions of the suggested service. Finally, we are investigating how the functionality specified for an overall goal for the process could be used to influence the choice of suggested services based on their functionality. Goals, preconditions and effects may be specified using the Web Ontology Language (OWL) and the Semantic Web Rule Language (SWRL).

In our system, we use SAWSDL (although if preconditions/effects are desired, WSDL-S [18] extensions are utilized) as the description language for Web services. A SAWSDL file may have semantic annotations for inputs, outputs and functionality. For inputs and outputs, the schema specification often forms a tree structure with semantic concepts as its nodes and the <message><part> as the root. The similarity of the trees will be calculated using the matching algorithms provided by Lumina (http://lsdis.cs.uga.edu/projects/meteor-s/downloads/Lumina/) [19] that ranks candidate Web services. Lumina semantically ranks the Web services based on data semantics and functional semantics [20].

5.3. Data mediation

Data heterogeneity is one of the major problems encountered during Web service composition. For example, student (ID, name, birthday) and student (ID, name) are different data. Even though they have the same name, they have different properties. If the output of the first Web service is student (ID, name, birthday) and the input of the second Web service is student (ID, name) then, when we compose these two Web services together into a workflow, the problem of data heterogeneity emerges. Data mediation aims at solving this problem.

Many researchers have studied the problem of data mediation in general and in regard to Web services. So far, there are mainly two algorithms using SAWSDL for data mediation: top-down [17, 21] and bottom-up [2, 22].

For the top-down algorithm, the main idea is to traverse the target XML schema tree in a top-down manner, and try to fill in each node with the data in the source XML schema. The target will be the XML schema of the input message of the second Web service. The source will be the XML schema of the output message of the first Web service. The source message is first transformed into a format identified by ontology concepts through the lifting schema mapping in SAWSDL. This format is then transformed into the target message format through the lowering schema mapping in SAWSDL. The details of this algorithm are in [17]. In our case, we are challenged to include data formats provided by Galaxy. To meet this challenge, we have implemented a data mediator capable of mapping, for example, tabular formats to a semantic definition along the lines of an abbreviated SAWSDL specification.

The bottom-up algorithm mainly compares the semantic annotations of the leaf-level nodes in the XML schema tree between source message and target message. From the leaf-level node, it will then traverse the schema tree in a bottom-up manner to the top to compose an XPath expression. The XPath expressions will be used in the <copy> element of the BPEL file to map the source message to the target message. Figure 4 shows the structure of the data mediator.

Our implementation of bottom-up algorithm was finished and already used in WS-BioZard [2, 22]. It can run standalone or easily be plugged into another system, so we can plug it in Galaxy and use it here for data mediation during Web service composition. We are currently implementing the top-down approach and plan to compare it with the bottom-up approach.

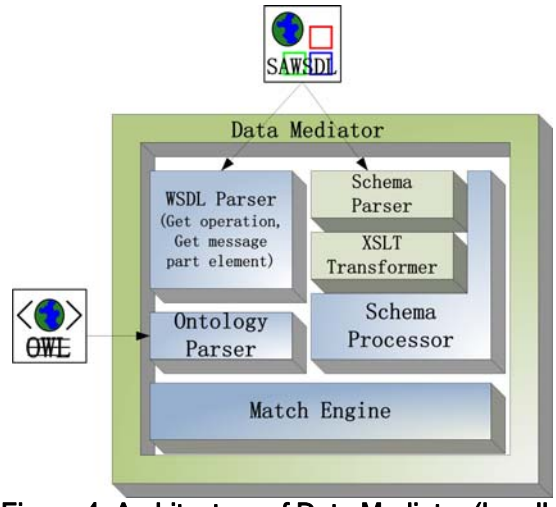


Figure 4. Architecture of Data Mediator (handle semantic date mediation)

6. Overview of implementation

Galaxy is implemented in Python using Pylons for its Web application framework. Therefore, Python is the basis of our implementation. Moreover, the ZSI Web Service library is used for helping with the integration of Web services into Galaxy.

6.1 System architecture

Our implementation adds two main modules to Galaxy, WS Adapter and WS Composer (Figure 5).

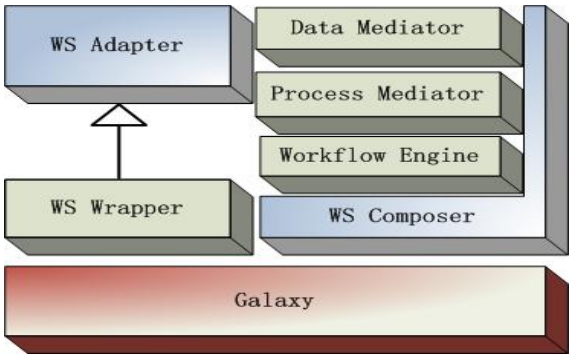


Figure 5. System Architecture

WS Adapter will automatically generate a wrapper for each Web service. It will also deal with the input/output and invocation of the Web service. In Section 3, we discussed in detail the functional modules of WS Adapter.

The WS Composer has three parts: the Workflow Engine, Process Mediator and Data Mediator. The Workflow Engine works to support our workflow model. It will save the Web service workflow as a BpelPy BPEL file and also have the capability to run the workflow. The Process Mediator implements our service ranking algorithm. It will suggest appropriate services to the user and deal with the process mediation mentioned in Section 5.2. The Data Mediator (Figure 4) has the responsibility of handling data mediation during Web service composition. Since our process mediation algorithm and data mediation algorithm are both based on the SAWSDL, ontology, etc., their implementations share some code, such as the code to parse ontology, parse SAWSDL, etc.

While we have modified/extended some of the internals of Galaxy, we kept the main Web interface. We also customized part of the user interface to support our extensions. For instance, some Web pages were created to support adding Web services by users. Some other Web pages were added to interact with users during data mediation and process mediation. All these Web pages were implemented with Mako and JavaScript.

7. Preliminary Evaluation

To test our extensions, we setup a Galaxy server on our own computer (demo site: http://128.192.251.200:8080) with all tests performed using the Mozilla Firefox browser. The evaluation tests were designed to show that we successfully adapted Galaxy to support semantic Web service composition. Specifically it includes tests of the following functionalities:

- Users can add Web services to our extended Galaxy.
- Users can compose Web services and the tools of Galaxy into a workflow with the help of semi-automatic process mediation and data mediation.

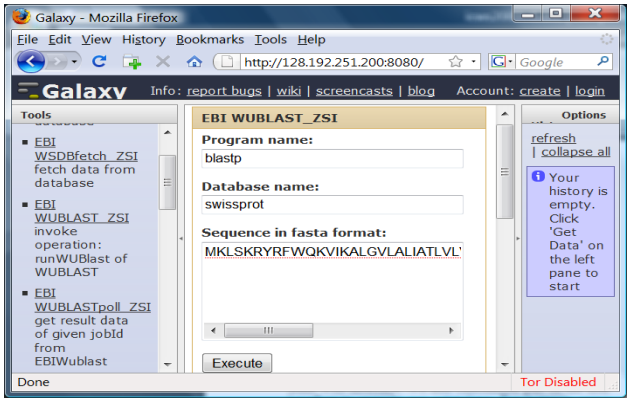


Figure 6. The runWUBlast operation of EBI WUBlast Web service

7.1 Test of adding Web services to Galaxy

To show that the user can add Web services to our extended Galaxy, we tested with two EBI Web services described in Section1.1.

After we typed the URLs in the user interface one at a time, Galaxy refreshed the interface and displayed all the newly added operations of the two Web services. We

invoked one operation: runWUBlast with the following inputs as shown in Figure 6. The response of job ID is returned. It showed that the Web service has been successfully added. This job ID can be fed into another operation, called poll, of the WUBlast Web service. The poll operation can retrieve the detailed result according to the given job ID.

7.2. Test of composing a workflow

This test is to compose a simple workflow (see the motivating workflow presented in Section 1.1) as shown in Figure 7. The inputs are shown on the right panel. After composition, a user can save the workflow into a BpelPy BPEL file or execute it.

As shown in Figure 9, the workflow actually converted the FASTA formatted output (Figure 8) of the first Web service into tabular format as the final result. Therefore, this test reaches the conclusion that our extended Galaxy based on our workflow model can support a workflow composed of Web services and Galaxy tools. Users can add Web services to Galaxy and compose a workflow with the added Web services. At present, we are working on semi-automatic data mediation and process mediation to facilitate Web service composition.

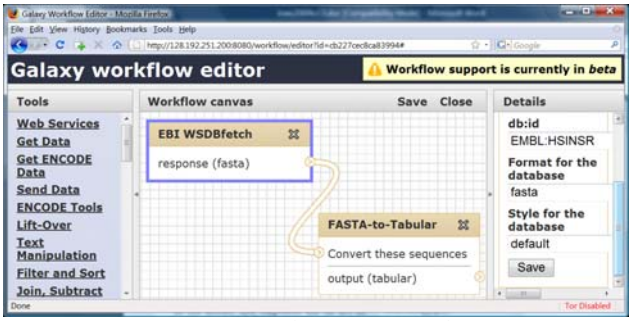


Figure 7. Sample workflow

Figure 8. FASTA formatted output of the first Web service of the sample workflow

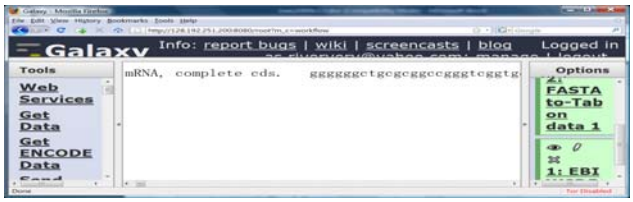


Figure 9. Final result of the workflow (in tabular format)

8. Conclusions and future work

We have added to Galaxy the ability to use Web services as well as a BPEL based workflow model. We are developing an online Web service composition tool for the biological domain. Our contributions include applying both top-down and bottom-up algorithms for data mediation and a DSL for a useful subset of BPEL in Python.

A unique aspect of our work is that we exploit available semantics, rather than mandating full semantic specifications, so our tool will work with whatever semantic annotations available. The simplest annotations are model references on an operation's inputs and outputs. This is enough to support bottom-up data mediation. If lifting and lowering schema mappings are also provided then top-down data mediation may be carried out. If the functionality of a Web service operation is semantically annotated, many irrelevant services may be ignored. Finally, although more difficult to provide, if annotations of an operation's preconditions are effects are provided, suggestions (or process mediation in general) can be further refined by checking logical consistency as well as compliance with goals.

These concepts of semantics, data mediation and process mediation will enable Galaxy to support semantic Web services composition and become a simple, easy to use process designer. These extensions to Galaxy will provide a much needed addition to Galaxy's biological analysis capabilities which allow non-technical users to create functional workflows.

Currently, we have completed the WS Adapter module. We are working on implementations related to semantics, specifically WS Composer including Data Mediator, Process Mediator and semantic workflow engine. We are also adapting our user interface to convey more semantic meaning and to be more user-friendly.

Acknowledgements

This project is funded in part by NIH R01 AI058515 awarded to Jessica C. Kissinger. J.M, J.C.K, E.K., D.B and R.W have been funded in whole or in part with Federal funds from the National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services, under Contract No. HHSN266200400037C. We would like to

thank James Taylor for his valuable insights into the Galaxy tool. We also express our gratitude to all our project members and ApiDB/EuPathDB project members for all their help, support, interest and valuable hints.

References

[1] Galaxy, http://galaxy.psu.edu/

[2] Z. Wang, J. A. Miller, J. C. Kissinger, R. Wang, D. Brewer and C. Aurrecoechea, "WS-BioZard: A Wizard for Composing Bioinformatics Web Services", Proceedings of SWF'08, in conjunction with SCC'08, Honolulu, Hawaii , Jul 2008, pp. 437-444.

[3] K. Sivashanmugam, K. Verma, A Sheth, J.A. Miller, "Adding Semantics to Web Services Standards", Proceedings of the ICWS, Las Vegas, Nevada, Jun 2003, pp. 395-401.

[4] K Sivashanmugam, JA Miller, AP Sheth, K Verma, "Framework for Semantic Web Process Composition", International Journal of Electronic Commerce, M.E. Sharpe, Winter 2004-05, Vol. 9(2), pp. 71-106.

[5] S. Dustdar, W. Schreiner, "A Survey on Web Service Composition", International Journal of Web and Grid Services, InderScience, 2005, Vol. 1(1), pp. 1-30.

[6] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau, "Automating DAML-S Web Services Composition Using SHOP2", 2nd International Semantic Web Conference (ISWC2003), Springer, Sanibel Island, Florida, 2003, Vol. 2870/2003, pp. 195-210.

[7] S. Thakkar (B) · J. L. Ambite · C. A. Knoblock, "Composing, Optimizing, and Executing Plans for Bioinformatics Web Services", The VLDB Journal, Trondheim, Springer, Norway, Sept 2005, Vol. 14(3), pp. 330–353.

[8] F. Casati, S. Ilnicki, and L. Jin, "Adaptive and Dynamic Service Composition in eFlow", Proceedings of 12th International Conference on Advanced Information Systems Engineering (CAiSE), Springer Stockholm, Sweden, Jun 2000, Vol. 1789/2000, pp.13-31.

[9] F. Casati, M. Sayal and M. Shan, "Developing E-Services for Composing E-Services", Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE), Springer, Interlaken, Switzerland, Jun 2001, Vol. 2068/2001, pp. 171-186.

[10] Y. Charif-Djebbar and N. Sabouret, "Dynamic Web Service Selection and Composition: An Approach Based on Agent Dialogues", Proceedings of the 2006, IEEE/WIC/ACM, Springer, Hong Kong, Dec 2006, Vol. 4294/2006, pp. 515-521.

[11] R. Hull, M. Benedikt, V. Christophides, J. Su, "E-Services: A Look Behind the Curtain", Proceeding of the 22th PODS, San Diego, USA, Jun 2003, Vol. 22, pp. 1-14.

[12] J. Rao and X. Su. "A Survey of Automated Web Service Composition Methods", Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, Jul 2004, Vol. 3387/2005, pp. 43-54.

[13] M Xu, J Chen, Y Peng, X Mei, and C Liu, "A Dynamic Semantic Association-Based Web Service Composition Method", Proceedings of the 2006 IEEE/WIC/ACM, IEEE, Hong Kong, Dec 2006, pp. 666-672.

[14] D Michael, P Rachel, W Mark, "Semi-automatic web service composition for the life sciences using the BioMoby semantic web framework", Journal of Biomedical Informatics, Elsevier Science, San Diego, USA, Oct 2008, Vol. 41(5), pp. 837-847

[15] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D.F. Ferguson (Editors), "Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More", Prentice Hall, USA, 2005.

[16] J. Farrell and H. Lausen, "Semantic Annotations for WSDL and XML Schema", http://www.w3.org/2002/ws/sawsdl/spec/2007, Aug 2007

[17] M. Nagarajan, K. Verma, A. P. Sheth and J. A. Miller, "Ontology Driven Data Mediation in Web Services", International Journal of Web Services Research (JWSR) , USA, Dec 2007, Vol. 4(4), pp. 104-126.

[18] R. Akkiraju, J. Farell, J. A. Miller, M. Nagarajan, A. Sheth and K. Verma, "Web Service Semantics - WSDL-S", Proceedings of the W3C Workshop on Frameworks for Semantics in Web Service (W3CW'05), Innsbruck, Austria, Jun 2005, pp. 1-5.

[19] K. Li, "Lumina: Using WSDL-S for Web Service Discovery," Master Thesis (M.S. in CS Degree), University of Georgia, December 2005

[20] S. Emani, "A Comparative Evaluation of Semantic Web Service Discovery: Algorithms and Engines," Master thesis (M.S. in CS Degree), LSDIS lab, University of Georgia, April, 2009

[21] Z. Wu, "Automatic Composition of Semantic Web Services using Process and Data Mediation", Technical Report, LSDIS lab, University of Georgia, Feb 2007

[22] Z. Wang, R. Wang, C. Aurrecoechea, D. Brewer, J. Miller, J. Kissinger, "Semi-automatic Composition of Web Service for Bioinformatics Domain", Technical Report, LSDIS lab, University of Georgia, May, 2008