Recent years have seen a tremendous rise in the number of bioinformatics tools available as Web services. Availability of tools as Web services, open-up possibilities for the creation of Web service compositions for performing complex data analysis that often involves multiple steps. Web services are usually developed independently of one another, making it difficult to construct Web service compositions. The Service Suggestion Engine (SSE) a system that uses semantically annotated Web services (using the SAWSDL mechanism), have been developed to help users in the construction of Web service compositions. SSE can provide suggestions for the next or previous step in the workflow. In order to support consistent annotation of bioinformatics Web services, the Ontology for Biomedical Investigations (OBI) has been extended to create OBI-WS, a Bioinformatics Web Service Ontology. The performance of SSE is evaluated against a consensus of domain experts.
BIOINFORMATICS WORKFLOW DESIGN
USING SEMANTICALLY ANNOTATED WEB SERVICES

by

ALOK AJIT DHAMANASKAR

B.E., University of Mumbai, India, 2009

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

MASTER OF SCIENCE

ATHENS, GEORGIA

2012
BIOINFORMATICS WORKFLOW DESIGN
USING SEMANTICALLY ANNOTATED WEB SERVICES

by

Alok Ajit Dhamanaskar

Approved:

Major Professors: Jessica C. Kissinger
John A. Miller

Committee: Krzysztof Kochut
Eileen Kraemer

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2012
BIOINFORMATICS WORKFLOW DESIGN
USING SEMANTICALLY ANNOTATED
WEB SERVICES

Alok Ajit Dhamanaskar

July 23, 2012
DEDICATION

To my parents for their endless love, support and encouragement.
ACKNOWLEDGEMENTS

Past two years have been and will remain one of the most enriching years in my life, and I cannot thank Dr. Jessica Kissinger and Dr. John Miller enough for this. Thank you for all the guidance, support and encouragement! I would like to thank Dr. Stoeckert for his inputs, insights and guidance for the project. I am very grateful to my committee members Dr. Kraemer and Dr. Kochut for their timely availability and suggestions. I would like to thank my colleagues Jie, Michael, Chaitanya, Frank and Arash for all their support and great work on the project. Also a special thanks to everyone in Jessie’s lab: Betsy, Jeremy, Brian, Susanne, Ganesh, Christina and Mark.
TABLE OF CONTENTS

ACKNOWLEDGEMENTS ......................................................... v
LIST OF TABLES ............................................................... viii
LIST OF FIGURES ............................................................... ix

CHAPTER

1 INTRODUCTION AND LITERATURE REVIEW ......................... 1

2 SERVICE SUGGESTION ENGINE: SIMPLIFIED CONSTRUCTION OF BIOINFORMATICS WORKFLOWS ................. 4
   2.1 Introduction ............................................................. 6
   2.2 Semantics for Web Services .......................................... 8
   2.3 Service Suggestion Engine .......................................... 15
   2.4 Interaction of SSE with Bioinformatics Workflow Composition Systems ............................................ 24
   2.5 Evaluation ............................................................. 26
   2.6 Related Work ........................................................... 32
   2.7 Conclusions and Future Work ...................................... 33

3 SUMMARY ................................................................. 36

REFERENCES ................................................................. 38
APPENDICES ................................................................. 45

Appendix A  Developers Guide ........................................... 46

Appendix B  SEMANTIC WEB SERVICE FRAMEWORKS: SAWSDL VS.

  OWL-S ................................................................. 48
  B.1 Introduction ......................................................... 48
  B.2 OWL-S: Semantic Markup for Web Services .................. 49
  B.3 Semantic Annotations for WSDL and XML Schema (SAWSDL) 49
  B.4 Observations ....................................................... 51
LIST OF TABLES

2.1 Description of sub-scores calculated by SSE ........................................ 16
2.2 Steps in the Workflow ................................................................. 27
LIST OF FIGURES

2.1 A Generic Model for Modelling Web Services . . . . . . . . . . . . . . . . . . 11
2.2 Branch of OBI-WS for Web Service Operations . . . . . . . . . . . . . . . 13
2.3 New Ontology Terms Added for Each Web Service Modeled . . . . . . . . 14
2.4 p-Homomorphism Input-Output Matching . . . . . . . . . . . . . . . . . . . 19
2.5 Extension to Galaxy UI to Invoke the Service Suggestion Engine (SSE) . . 25
2.6 Comparison between Forward Suggestions for Path-Based and p-Homomorphism
   Input-Output Matching Algorithms at Different Levels of Annotation . . . . . 29
2.7 Comparison Between Path-Based and pHomomorphism Input-Output Match-
   ing Algorithms for Forward, Backward and Bidirectional Suggestions . . . . 31
B.1 The Top Level Hierarchy for OWL-S . . . . . . . . . . . . . . . . . . . . . 50
B.2 SAWSDL Schema matching Mechanism (Lifting and Lowering) for Data Me-
   diation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
Chapter 1

INTRODUCTION AND LITERATURE REVIEW

Web services are becoming a popular, even preferred way to provide and support applications or services over the Web, thus exposing the vast amounts of data that are locked up in databases (that previously were accessible only through custom APIs). For example, in the bioinformatics domain, BioCatalogue [1] alone has 2293 Web services. Web services are Web applications that have well-defined interfaces which facilitate their programmatic invocation by other applications. Exposing either data or applications as Web services establishes a standard, uniform way in which anyone, or any system, can access these resources irrespective of the programming language used or any other implementational detail. This relieves the developers of the burden of maintaining individual interfaces for each of the systems, either internal or external, that needs to interact with it. Web services fall into two broad categories: SOAP and REpresentational State Transfer or REST Web services. SOAP and REST Web services primarily differ in the way they exchange input-output messages over the Web. SOAP Web services exchange messages using the Simple Object Access Protocol over a standard internet transport protocol typically HTTP, while REST Web services directly
use the HTTP methods to send and receive messages. A Web service interface is described using a Web service description document. SOAP Web services are described using a WSDL document while REST Web services can be described using either a WSDL or a WADL document. Both Web service Description Language (WSDL)¹ and Web Application Development Language (WADL)² are XML-based languages that specify the operations supported, their inputs and outputs and the accessibility information for a Web service. SOAP defines a messaging framework using XML. Defining XML as a standard language for exchanging messages over the Web has created a foundation for Web service inter-operability.

Analysis of bioinformatics data usually requires the coordinated use of multiple tools. These tools can be “strung” together to create a reusable workflow. This process is called composition. With the increasing availability of bioinformatics tools as Web services, Web service compositions are becoming increasing popular for complex, repetitive data analysis. Various systems have been developed to provide graphical interfaces to facilitate creation of workflows from Web services or tools. Galaxy [2] and Taverna [3] are two popular tools that provide a GUI for creating workflows.

Taverna is an open source Workflow Management System for the design and execution of scientific workflows, that is available as a desktop application. Taverna is integrated with BioCatalogue, and supports the invocation of Web services and their use in workflows. Galaxy is a Web-based platform for bioinformatics data analysis. Galaxy comes bundled with its own set of tools but it also allows the addition of new tools with some amount of programming. It supports independent usage of the tools as well as construction of analysis pipe-lines (workflows) using an easy to use, yahoo pipes like, workflow designer. The tool developed in [4] facilitates addition of Web services as tools to Galaxy. This addition makes Web service composition possible in Galaxy.

¹WSDL: http://www.w3.org/TR/wsd1
²WADL: http://www.w3.org/Submission/wadl/
Despite of the availability of various workflow designer tools, construction of Web service compositions still remains a difficult task to achieve for a number of reasons. Firstly, Web services are generally developed independently of one another without any considerations of interoperability. Secondly, the documentation of a Web service in the form of a WSDL document lacks a standard naming convention. The increasing availability of Web services only adds to the problem of choosing the right Web service for the job that is also input-output compatible with other operations in the workflow. Service Suggestion Engine (SSE) has been developed with the purpose of serving as a general purpose tool to provide assistance in the workflow creation process. SSE uses semantically annotated Web services (annotated using the SAWSDL mechanism) [5] to make suggestions for the next, previous or intermediate steps in a workflow.

The work in this thesis is divided into two main categories. First, OBI-WS, a bioinformatics Web services ontology, was created to support consistent annotation of bioinformatics Web services. As a part of this effort a systematic methodology was devised and adopted for modeling Web services that led to creation of 116 terms to support the annotation of 19 different Web services. Second, work was performed to improve and extend the previous work on suggestions for semi-automatic workflow composition [6, 7].
Chapter 2

SERVICE SUGGESTION ENGINE: SIMPLIFIED CONSTRUCTION OF BIOINFORMATICS WORKFLOWS

\footnote{Alok Dhamanaskar, Michael Cotterell, Jie Zheng, Christian Stoeckert, Jr., Jessica C. Kissinger, and John A. Miller. To be submitted to Bioinformatics.}
Abstract

In recent years, in the Bioinformatics domain, Web services are becoming a preferred way to offer applications or services over the Web and also to expose the vast amounts of data that is locked up in databases. This opens-up possibilities for creation of Web service compositions to perform bioinformatics data analysis. However, creation of Web service compositions requires selection of the appropriate Web service operations from the numerous available Web services (BioCatalogue alone has 2293 Web services), that provide the desired functionality and are also input-output compatible with the operations currently in the workflow. To help the user with the design of Web service compositions, we have developed the Service Suggestion Engine (SSE) to address a variety of issues involved in Web service composition. The SSE makes use of semantically annotated Web services to provide suggestions to the user for the next or previous step in a workflow. We have extended the Ontology for Biomedical Investigation (OBI) to create OBI-WS, a bioinformatics Web services ontology to provide the necessary vocabulary for annotation of Web services in this domain. SSE has been evaluated in a real world bioinformatics workflow construction scenario. We have considered a commonly encountered scenario of creating a workflow to discover more information about a particular protein sequence and its evolutionary relationship with other protein sequences. The SSE has shown promising results with an average $F_2$-measure of little over 0.7 when suggesting the next or previous step in the workflow from a total of 101 possible Web service operations. The results, in addition to vouching for the usability of SSE, provide a means to evaluate the quality of the semantics developed through OBI-WS for describing bioinformatics Web services. The current release for OBI-WS (OBI extended Web services ontology) can be accessed from NCBO BioPortal\(^2\).

\(^2\)http://bioportal.bioontology.org/ontologies/3119
2.1 Introduction

The post-genomic era in the bioinformatics world has been characterized by an exponential rise in available data. The databases reported by the Nucleic Acid Research Journal [8] have increased to 1380 in 2011 from 218 databases in 2001. This availability of enormous amounts of data presents opportunities for bioinformatics analysis to mine this data for interesting insights. However, the ever-increasing availability of data is accompanied by issues such as the data, being physically distributed, available in different formats and exposed through varied interfaces. Web services are becoming a preferred way to expose this data over the Web. This, coupled with the increasing availability of bioinformatics tools wrapped as Web services, have made in silico workflows a preferred means of facilitating analysis of bioinformatics data from heterogeneous resources. “Workflows are thus becoming a paradigm for enabling science on a large scale by managing data preparation and analysis pipelines, as well as the preferred vehicle for computational knowledge extraction” [9].

Performing in silico experiments using workflows can save bioinformaticians and biologists substantial amounts of time by automating routine tasks for easy reuse. Also, once a workflow is constructed it can benefit multiple users with little or no modification, thus saving hours of repetitive work. Different workflow composition platforms provide the ability to store and share workflows. Workflows thus support ”Reproducible Computational Research” by producing results that are easily reproducible and verifiable. In order to facilitate construction and execution of bioinformatics workflows tools such as Taverna [3], Galaxy [2] and Kepler [10] have been developed. However, the availability of tools as Web services which provide flexibility with respect to their use and invocation also poses challenges like (1) finding the right Web Service operation for the job, (2) finding operations that are input-output compatible and (3) connecting the right inputs to correct outputs.

Web services are often developed independently of one another and their documentation
in the form of a Web service description document, is devoid of any standard naming convention. For instance the same input **Expectation value** can be referred to as **e-val** or **exp** or even worse the operation that performs multiple sequence alignment being referred to as **run**. Semantic annotations from ontologies can help in terms of providing formalized Web accessible documentation to describe the operations found in Web services.

The Service Suggestion Engine (SSE), a product of our work, [6, 7, 11] is developed to serve as a general purpose tool to provide assistance in the creation of workflows. It can work alongside existing bioinformatics workflow composition platforms such as Galaxy and Taverna and help a bioinformatician or a biologist who wants to create a workflow. The SSE provides suggestions for the next or previous step in the form of a ranked list of available Web service operations (referred to as **candidateOps**). The **candidateOps** are ranked depending upon the input-output compatibility of the **candidateOp** with the operations that are currently in the workflow (**WorkflowOps**) and how well the functionality of the **candidateOps** aligns with the desired functionality. SSE implements different algorithms to determine the input-output mapping that takes into account the XML input-output structure in varying extents. SSE uses semantically annotated Web services, annotated using the, Semantic Annotations for WSDL (SAWSDL) [5] standard.

We have developed OBI-WS, a bioinformatics Web services ontology that uses the Ontology for Biomedical Investigation (OBI) as a base ontology. A systematic methodology was adapted in extending the Ontology of Biomedical Investigation (OBI) to create OBI-WS. Section 2.2 summarises the process followed for the creation of required semantics in the form of OBI-WS and a summary of terms that were added. Section 2.3 describes the details of SSE in terms of calculation of different sub-scores and some of the data matching algorithms it implements. Section 2.4 briefly talks about the extensions provided for invoking SSE from Galaxy and a possibility of doing the same for Taverna. The evaluation of SSE in a real world workflow creation scenario is described in section 2.5. Sections 2.6 and 2.7 talk
about the related work and conclusions, respectively.

2.2 Semantics for Web Services

Web services provide a means to expose the functionality provided by an application over the Web, defined using standard interfaces. Web services fall into two categories of SOAP Web services and REpresentational State Transfer (REST) Web services. SOAP and REST Web services primarily differ in the way input-output messages are exchanged over the Web. SOAP Web services exchange messages using the Simple Object Access Protocol over a standard Internet transport protocol (typically HTTP), while REST Web services directly use the HTTP methods to send and receive messages.

A Web service is described using a Web service description document. SOAP Web services are described using a Web Service Description Language (WSDL) document and REST Web services can be described using either a WSDL 2.0 or a Web Application Development Language (WADL) document. Both WSDL and WADL are XML-based languages for describing Web service in terms of the operations supported, their inputs and outputs along with the accessibility information for a Web service. SOAP defines a messaging framework using XML. Defining a common language for exchanging messages over the Web creates a foundation for Web service inter-operability. The information provided in the WSDL document ‘might’ be sufficient when it has to be consumed by an application, assuming that the user understands what the input, output and operation names are referring to. However, the scenario gets complicated when a user has to choose from numerous of available Web services or a system has to provide suggestions for a Web service composition. Some sort of additional documentation for all the relevant WSDL elements that can unambiguously describe what the element (input, output or operation) refers to, is Web accessible and describes how that term is related to other concepts in the same domain is desired.
Ontologies, come close to providing these desired features thus making it ideal for annotating Web services. Standardized annotations of a Web service include the semantics for the input, output and functionality of each of the service’s operations. Semantic Annotations for WSDL and XML Schema (SAWSDL) is a W3C Recommendation that provides a simple mechanism for adding semantics to Web services. All of the concepts in the ontology have a unique Web accessible URI. Semantic annotations are references or Uniform Resource Identifiers (URIs), added to a WSDL document using the \texttt{modelReference} extension attribute provided by SAWSDL that point to the relevant ontology concept to be used for annotation.

2.2.1 OBI-WS, Bioinformatics Web Services Ontology

Bioinformatics Web services are used to analyze biological and biomedical data and hence, need domain-specific terms for their annotation. For example, the terms nucleic acid or protein sequence, are as important as terms used to describe the data format, statistical significance and analysis parameters. Many bio-ontologies have been developed under the Open Biomedical Ontologies (OBO) [12] umbrella and used widely in the biological field, such as the Sequence Ontology [13] and Gene Ontology [14]. OBO compliant ontologies share a common upper level ontology, the Basic Formal Ontology (BFO) [15], and a common set of relations, the Relation Ontology (RO). The Ontology for Biomedical Investigations [16], an OBO Foundry candidate ontology, is being developed to address the need for consistent description of all aspects (including data analysis) of biological and clinical investigations across disciplines and is written in OWL-DL. OBI is a process oriented ontology that models a process with input, output and objective specifications and is suitable for supporting Web service annotations [17]. Additionally, OBI contains high-level terms such as planned process, service, algorithm, data transformation and data format specifications that provide a foundation which can be extended to add terms to support the annotation of Web services. OBI-WS (OBI extended Web services ontology) is the outcome of our efforts towards
extending OBI to support annotation of bioinformatics Web services.

2.2.2 Methodology for Extending OBI

Ontologies used for annotations should provide terms that correspond to key aspects of a Web service description, i.e., the operations and their inputs and outputs. We have devised and adopted a systematic methodology for extending the OBI ontology to create OBI-WS that can support annotations of bioinformatics Web services [17]. We model the Web services along the lines of a generic model that we have created. We have created a generic model (see figure 2.1), that indicates the higher level terms in OBI-WS to be used as super classes for adding new Web service operations (OBIWS: Web service execution), the functionality achieved by the Web service operation (OBI:objective specification) and terms for describing its inputs and outputs (IAO:information content entity). Using the generic model as the basis we individually model Web services to incrementally add terms to the ontology. We have focused our efforts on Web services that perform different kinds of sequence analysis including Web services for sequence similarity searches, multiple sequence alignment, protein functional analysis and phylogenetic analysis.

For every Web service that we model, we start by populating an spread sheet template we have created to consistently record the required information. For each WSDL element (input, output or operation) recorded in the spread sheet we determine if a suitable term that can be used for annotation already exists in OBI or OBI-WS. If no suitable term is found, we search the National Center for Biomedical Ontology (NCBO) BioPortal\(^3\) to see if a suitable term exists in any of the OBO compliant ontologies. If it does, we import it into OBI WS using the Minimal Information to Reference External Ontology Terms (MIREOT)\(^4\). This is

\(^3\)NCBO BioPortal: http://bioportal.bioontology.org/
\(^4\)MIREOT: http://obi-ontology.org/page/MIREOT
where the advantage of extending OBI, which is OBO compliant, comes in. The EMBRACE Data and Methods (EDAM) Ontology [18] has been developed to describe, at a coarse level, many bioinformatics databases, data and tools. If the terms we need are defined in EDAM we try to reuse the definition provided by them.

The absence of a suitable term justifies the need for a new term to be added to OBI-WS. Depending upon the term and what it describes (operation, objective or input/output) we try to determine a super class in the ontology under which the new term can be added. In the spread sheet for the Web service we record the label, super class, definition source and definition for the term. The terms are defined in consultation with domain experts whenever needed. When defining the label and definition, every possible effort is made to ensure that the term aptly describes the WSDL element but at the same time is general enough to be
used by other Web services. Once the spreadsheet is completely populated such that for every term in the WSDL document there either exists a term in OBI-WS or a new term has been proposed the newly-proposed terms are added to the development version for OBI-WS, using WebProtege [19]. WebProtege is a Web-based collaborative ontology editor that allows multiple users to simultaneously edit an ontology and add notes for discussions. Once the new term is added to the ontology we add the logical definition for the term/ontological concept in terms of restrictions on the properties. Typically, restrictions added on the properties OBI:has_specified_input and OBI:has_specified_output for inputs and outputs that are specific to the operation and OBI:achieves_planned_objective for the objective specification achieved by the Web service operation. Figure 2.2 shows the branch of the OBI-WS under which Web service executions are added. After the restrictions are added, reasoning is performed on the ontology to check for consistency of the added terms, as well as to infer the correct placements of the terms in the ontology’s hierarchy using HermiT [20]. Once the correct placement of the term and the consistency of the ontology is ensured, we assign a permanent id to the term from the range starting with OBIws_0000001. This is continued for all the terms proposed for a particular WSDL.

### 2.2.3 Summary of Terms Added

Using this methodology we have created terms for the description 101 Web service operations from 19 different Web services resulting in the creation of 116 new terms. This includes seven terms reused from EDAM and two terms imported from the Sequence Ontology(SO). The spreadsheets for all the modeled Web services as well as the latest version of the ontology can be found here\(^5\). As discussed earlier, the Web services modeled cover different types of sequence analysis services including, but not limited to, WU-BLAST, NCBI BLAST,

\(^5\)OBI WS: [https://code.google.com/p/obi-webservice/](https://code.google.com/p/obi-webservice/)
ClustalW2, T-Coffee, InterProScan, SignalP, wsPhylipProtdist and wsPhylipNeighbor. We have observed that as more and more Web services in the domain are modeled the need to create new terms reduces, see figure 2.3. This will make annotation of newer Web services in the same domain much quicker and easier.

We have made available two different versions of OBI-WS

1. OBI-WS Development version, which contains all the terms in the OBI ontology plus the terms for annotation of Web services and

2. OBI-WS Release version, which is a lighter version that contains only the Web service terms along with the related OBI higher level-terms. This version due to its smaller size is faster to load and reason and hence is more suitable to be used with applications.

We use Ontodog\(^6\) in order to create the OBI-WS Release version. Ontodog is an easy to use

\(^6\)http://ontodog.hegroup.org/
Web-based ontology view generator that when given a source ontology and a list of terms desired in the view generates a subset (view) of the ontology that has the desired terms as well as all the related terms.

Adding annotations to Web services involves adding references to ontology terms in the WSDL document. Annotating Web services can be quite painstaking. The user first needs to find the correct ontology term to use for annotation and then they need to edit the WSDL document (an XML file) to add the annotation to the appropriate item, without missing any tags or imports required for SAWSDL. We have created a tool, RadiantWeb [21] to partially automate this process. RadiantWeb works at different levels of automation and can provide recommendations for terms from the ontology that can be used for annotation of relevant WSDL elements (inputs, outputs and operations). Apart from providing recommendations, it provides a simple drag and drop based interface for manual annotation of WSDL documents.
2.3 Service Suggestion Engine

The SSE can assist in the process of workflow creation by suggesting Web service operations for the next or previous step. Two important aspects that are considered when selecting an operation to be added to a workflow are that (1) the new operation should be input-output compatible with the operation(s) currently in the workflow (WorkflowOps) and (2) the functionality that it serves/provides should align with what the user is intending to do. SSE takes into account both of these aspects when scoring the available operations (candidateOps) in terms of an input-output compatibility sub-scores and an objective specification compliance sub-scores.

The input-output matching algorithms, calculate a input-output compatibility sub-score depending on the similarity between the input-output XML structures. Both input-output matching algorithms described in the sections below distinguish between the required and optional inputs, weighing the optional inputs lower than the required ones.

The objective specification for an operation indicates the purpose or the overall functionality of the operation. It is not the detailed name or signature for the operation as it is unlikely that a workflow designer would know these specifics and if they did the SSE may not be needed. The operations are annotated by a concept in the ontology that indicates its objective specification, i.e., the goal/functionality that is achieved by the operation. Compliance here refers to how close the desired functionality supplied by the user in the form of keywords or an ontology concept is to the objective specification of the operation.

\[ S_t = \frac{w \cdot s}{w \cdot a} \] (2.1)

The scores for the candidateOps are calculated as in equation 2.1, where s is a vector of sub-scores listed in table 2.1. a is a vector of Boolean values, indicating whether a sub-score is to be used. For example, if an operation is not annotated with an objective specification
then $a_4 = 0$.

<table>
<thead>
<tr>
<th>Sub-score</th>
<th>Weight</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{io}^{sym}$</td>
<td>$(1 - \sigma)(1 - \varphi)$</td>
<td>input-output syntactic sub-score</td>
</tr>
<tr>
<td>$S_{io}^{sem}$</td>
<td>$\sigma(1 - \varphi)$</td>
<td>input-output semantic sub-score</td>
</tr>
<tr>
<td>$S_{obj}^{sym}$</td>
<td>$(1 - \sigma)\varphi$</td>
<td>objective specification syntactic sub-score</td>
</tr>
<tr>
<td>$S_{obj}^{sem}$</td>
<td>$\sigma\varphi$</td>
<td>objective specification semantic sub-score</td>
</tr>
</tbody>
</table>

Table 2.1: Description of sub-scores calculated by SSE

In table 2.1, $\sigma$ represents the preference to be given for semantics over syntax, e.g., if $\sigma$ is 0.7, we weigh semantics at 0.7 and syntax at 0.3. Similarly, $\varphi$ represents the preference to be given for objective specification compliance over input-output compatibility, e.g., if $\varphi$ is 0.4, then we weigh compliance at 0.4 and compatibility at 0.6. If a clear naming convention is followed and the the ontology is not thoroughly specified, then it would make sense to reduce the value of $\sigma$. With these values for $\sigma$ and $\varphi$ the weight vector $\mathbf{w}$ becomes $(0.18, 0.42, 0.12, 0.28)$.

### 2.3.1 Algorithms for Input-Output Compatibility

To examine the input-output compatibility, we need to model how types are specified in the WSDL documents. In WSDL 2.0, types are defined entirely using XSD, while in WSDL 1.1, WSDL itself includes type constructors. In either cases, most of the complexity is from the XSD specification [22]. There have been a few papers that have modeled XSD using graphs: [23] use directed graphs, [24, 25] use directed acyclic graphs (DAG) and [26, 27] use trees. Currently, we restrict our attention to DAGs and trees. In particular, we model inputs/outputs using an input-output Directed Acyclic Graph (DAG) ($G_{io}$). $G_{io}$ is defined as a vertex labelled DAG, where the each vertex $v \in V$, in the DAG can be an XSD element, a complex type or a simple type. This allows us to retrieve all the necessary information specified by their attributes which is used in the calculation of input-output compatibility.
sub-scores. Currently, we make use of the attributes: xsd:name, xsd:type, xsd:nillable, xsd:minOccurs and sawsdl:modelReference.

\[ G_{io} = \langle V, E, l : V \rightarrow M \rangle \]

- \( E \subset V \times V \)
- \( G_{io} \) is acyclic: Denoting a path \( p \) as a sequence of vertices connected by edges \((v_1 \rightarrow v_2 \rightarrow v_3 \ldots v_{n-1} \rightarrow v_n)\), the graph is acyclic if \( \nexists \) a path where a vertex appear twice

- \( M = \{\text{name}, \text{type}, \text{required}, \text{modelReference}\} \)

where \( \text{required} \) equals xsd:nillable or minOccurs = 0.

Modeling the inputs and outputs as DAGs transforms the I-O matching problem into a graph pattern matching problem. SSE currently supports path-based and p-Homomorphism input-output matching algorithms for the calculation of input-output matching sub-scores.

**p-Homomorphism Input-Output Matching Algorithm**

Graph homomorphism is a structure preserving mapping between two graphs, which maps every vertex in one graph, to some vertex in other graph. In input-output matching a maximum possible match between the vertices of \( G_{io} \) is desired such that their structure is preserved. Hence determining if the DAG of the input is homomorphic to the output DAG of the previous operation should ideally suffice. However, the following issues make applicability of homomorphism to the problem of input-output matching difficult:

- Web services are created independently of one another and hence the chances of finding an exact homomorphism mapping between inputs and output DAGs is unlikely in many cases.
The vertices in $G_{io}$ are represented by labels and concepts in the Ontology and hence matching between the vertices needs to consider the similarity between the vertices as well.

In general, the graph homomorphism problem is NP-Hard [28], but if the graphs’ tree-widths are bounded, then the problem is in P (trees have a tree-width of one).

The concept, p-Homomorphism, introduced in [29] extends the notion of homomorphism by accounting for similarity between the vertices and provides a measure for quantifying the similarity between graphs when they are not exactly homomorphic to one another (see figure 2.4). Given two vertex-labelled DAGs, $G_1(V_1, E_1, l_1), G_2(V_2, E_2, l_2)$, a threshold $\tau$ and similarity function $\rho: \mathcal{M} \times \mathcal{M} \rightarrow [0, 1]$, p-homomorphism is a mapping function $h : V_1 \rightarrow V_2$ such that,

$$\forall u, v \in V_1, \rho(u, f(u)) > t \land \rho(v, f(v)) > t \land (u, v) \in E_1 \implies (f(u), f(v)) \in E_2.$$ 

Our implementation is based on the algorithm maximumCardinality in [29], which is an approximation algorithm. It takes in as input, input and output DAGs ($G_{io}$) as matrices, a similarity score matrix $\text{SimMat}$ (which corresponds to $\rho : \mathcal{M} \times \mathcal{M} \rightarrow [0, 1]$), the vertex weight function $VW : V_1 \rightarrow [0, 1]$ and threshold $\tau$. The similarity score matrix stores the pre-computed similarity score between every vertex $u \in V_1$ and $u' \in V_2$. As discussed before every vertex in a DAG is an XSD element which might be annotated with an ontology concept. The Similarity score between the nodes is calculated as,

$$\text{SimMat}(u, u') = (1-\varphi)(\sigma \text{SemSim}(\text{concept}_u, \text{concept}_{u'}) + (1-\sigma) \text{SynSim}(\text{label}_u, \text{label}_{u'}))$$

(2.2)

where the nodes $u$ and $u' = h(u)$ are annotated with a concepts $\text{concept}_u$ and $\text{concept}_{u'}$ and have the XSD element names $\text{label}_u$ and $\text{label}_{u'}$, respectively. The score for semantic similarity $\text{SemSim}(\text{concept}_u, \text{concept}_{u'})$ is provided by the Concept Similarity measure, described in section 2.3.3. The syntactic similarity between the vertices is computed using a string
similarity metric. Threshold $\tau$ specifies the minimum similarity score value for a mapping of vertices $u \rightarrow u'$ to be considered an acceptable match.

Some of the inputs for a Web service can be optional, meaning it is not mandatory to supply these inputs for a successful execution of the Web service. This is specified in the WSDL document using the attributes `nillable` and `minOcurr`. Since many Bioinformatics Web services have a large number of optional parameters, to fine tune the execution, it is important to weigh the optional inputs, or optional branches in the input DAG lower than the required ones. $VW(u)$ specifies the vertex weight for vertex $u$ in the input DAG. Weights for the vertices are calculated as follows: For every vertex $u$, where $\text{outdegree}(u) = 0$ (i.e., $u$ is a leaf vertex)

$$VW(u) = 0.2 \text{ if leaf vertex is optional, and 1.0 if otherwise} \quad (2.3)$$

For every vertex $u$ where $\text{outdegree}(u) > 0$ (i.e., $u$ is a non-leaf vertex), the weight is calculated as a generalized mean (with exponent of three) of the weights of the child vertices.
\[ VW(u) = \frac{1}{n} \left( \sum_{i=1}^{n} VW(v_i)^3 \right)^{1/3} \], where \( v_i \) is a child vertex of \( u \). \hspace{1cm} (2.4)

The algorithm attempts to maximize the mapping between vertices in input and output DAGs. Among the many data structure that the algorithm maintains during the execution, it maintains a list of acceptable matches for a vertex in the input graph to the vertex in the output graph and is of form \( u : \{v_1, v_2, ..., v_n\} \) for every vertex \( u \) in the input DAG to vertex \( v_i \) in the output DAG. The algorithm makes a greedy choice for a match \((u, v_i^*)\), which has the highest similarity score from the list of acceptable matches. Once the choice for a match is made, the algorithm proceeds to calculate two different mapping functions \( h \), (1) considering \((u, v_i^*)\) as a match in the final mapping in \( h \) and (2) without considering the match \((u, v_i^*)\) in the final mapping. The match that does not yield a mapping in \( h \) with the maximum match is discarded. With every iteration, the list of acceptable matches is trimmed to remove greedy choices that did not yield a maximum mapping. The details of the algorithm can be found in [29].

Once a \( p \)-homomorphism mapping is constructed, the \textit{input-output compatibility} sub-scores between the two DAGs are calculated as in the equations 2.5 and 2.6. The similarity scores for each match \((u, u')\) in the mapping \( h \) is multiplied by the vertex weight of \( VW(u) \).

\[
S_{io}^{sem} = \sigma (1 - \varphi) \frac{\sum_{(u,u') \in h} VW(u) \cdot SemSim(\text{concept}_u, \text{concept}_{u'})}{\sum_{v \in V_1} VW(v)} \tag{2.5}
\]

\[
S_{io}^{syn} = (1 - \sigma) (1 - \varphi) \frac{\sum_{(u,u') \in h} VW(u) \cdot SynSim(\text{label}_u, \text{label}_{u'})}{\sum_{v \in V_1} VW(v)} \tag{2.6}
\]
Path-Based Input-Output Matching

p-Homomorphism matching is a structure preserving input-output matching algorithm. Path-based matching is somewhat more flexible in that regard, but still considers some structural information. Path-based matching decomposes the input-output DAGs into individual paths. The algorithm tries to find the best matching path for each of the input paths of the candidateOp. Path-based matching calculates the input-output compatibility score based on how well the paths in the two DAGs can be matched with each other. The similarity score for vertices in the path are calculated in the same way as for p-Homomorphism matching, given by equation 2.2. The evaluations in [11] have shown path-based matching to perform well. The details of the algorithm can be found in [6].

2.3.2 Objective Specification Compliance Matching

As discussed before, the functionality that an operation serves should align with what the user is intending to do. SSE expects that the operations in the WSDL document are annotated with an ontology concept that represents the functionality achieved by the operations. The generic model model for Web services (figure 2.1 in section 2.2.2 recommends that an operation be annotated with a concept that is a sub-class of IAO:objective specification. The user can provide the desired functionality for the next step as an input to the SSE. He can choose to describe the desired functionality using keywords or as a concept in the ontology that he feels closely describes the functionality desired. If a concept is provided, the Objective Specification Compliance score is calculated as Concept Similarity between the concept provided by the user and the concept with which the Web service operation is annotated. If the semantics are not provided, the score is calculated using string similarity metric. If a desired functionality is not specified by the user, only input-output matching sub-scores are considered for suggesting operations, in which case the vector \( a \) becomes \((1,1,0,0)\).
2.3.3 Concept Similarity

The *input-output compatibility* and *objective specification compliance* matching algorithms use the concept similarity measure to compute a similarity score that indicates how close two concepts in an ontology are. Ontologies provide a way for representing knowledge, which is modeled as concepts and relationships between them. The relationships are defined in the ontology using object properties and restrictions. One of the advantages of using annotations from ontologies is that, in cases where either the input-outputs or the objective specification are not annotated with the same concept, calculation of a similarity score between them can still provide valuable insight into how close they are. The calculation of concept similarity $CS$ tries to account for all possible aspects of an ontology concept, namely, its label, class name, definition, the properties and restrictions defined for the concept and the relative position of the concept in the ontology hierarchy.

Concept similarity between two concepts $C_I$ and $C_O$, $CS(C_I, C_O)$ is calculated as in equation 2.7 [11], which considers syntactic, coverage and property similarity between two concepts.

$$CS(C_I, C_O) = \omega_1 \cdot Syntactic_{sim}(C_I, C_O) + \omega_2 \cdot Coverage_{sim}(C_I, C_O) + \omega_3 \cdot Property_{sim}(C_I, C_O)$$

(2.7)

Syntactic similarity sub-score takes into account the similarity between labels, class names and definitions for the two concepts and is calculated using Levenshtein’s distance and the Dice algorithm. The coverage similarity sub-score is an indication of the proximity of the two concepts in the ontology hierarchy. Coverage similarity is calculated as in equation 2.8.
\[ Coverage_{\text{sim}}(C_I, C_O) = \begin{cases} 
1, & C_I = C_O \\
e^{-\lambda_1 x}, & C_I = \text{ancestor}_x(C_O) \\
e^{-\lambda_2 x}, & C_O = \text{ancestor}_x(C_I) \\
e^{-\lambda_3 x}, & \text{for all other cases} 
\end{cases} \quad (2.8) \]

where \( x \) is the length of the path between concept \( C_I \) and concept \( C_O \) in the ontology hierarchy and \( \lambda \) is the adjustable rate of decay such that \( \lambda_1 < \lambda_2 < \lambda_3 \), with defaults as 0.25, 0.5 and 0.75 for \( \lambda_1, \lambda_2 \) and \( \lambda_3 \) respectively. [6].

The property similarity sub-score considers the properties defined for the individual concepts and restrictions on them. In presence of a restriction on a property, the range for the property is adjusted to reflect the range that is specified by the restriction. Currently, we do not distinguish between the \texttt{owl:someValuesFrom} and \texttt{owl:allValuesFrom} restrictions. Calculation of \( PropertySim(C_I, C_O) \) involves computing a matrix \( PropertyMatch \) that stores match scores between every pair of properties \( (p_1, p_2) \) \( \forall p_1 \in P_I \) and \( p_2 \in P_O \), where \( P_I \) is a set of properties for concept \( C_I \) and \( P_O \) is the set of properties for concept \( C_O \) which are calculated as in equation 2.9. Once the matrix is computed we use the Hungarian algorithm [30] to obtain an optimal assignment between the properties of two concepts which provides for the property similarity score, \( Property_{\text{sim}}(C_I, C_O) \).

\[ PropertyMatch(p_1, p_2) = \eta_1 \cdot PropSyntacticSim(p_1, p_2) + \eta_2 \cdot RangeSim(p_1, p_2) \quad (2.9) \]

We have developed ConceptSimilarity an independent module, so that it can be used by other algorithms to facilitate Semantic Web service discovery [31].
2.4 Interaction of SSE with Bioinformatics Workflow Composition Systems

The Service Suggestion Engine is developed with the purpose of providing help in the process of workflow creation and is meant to work alongside bioinformatics workflow composition systems like Galaxy and Taverna. The SSE is built to work with different levels of annotations, i.e., it can suggest Web service operations, whose inputs and outputs are either partially annotated or not annotated at all. In such cases SSE makes use of the element names in the WSDL, XSD:types and WSDL documentation to find matching Web service operations. Absence of semantics is penalized to some extent and string metrics, Levenshtein’s distance and the Dice algorithm are used to find the similarity between the XML inputs and outputs. SSE is not expected to work as well without semantics as it would with semantics. We explore the performance of SSE with and without semantics in the Evaluation section.

In order to facilitate the use of SSE with different bioinformatics workflow composition systems with minimum coupling, we have made SSE available as a RESTful Web service. It takes as input:

- workflowOps, a list of Operations currently in the workflow (Operation name and WSDL/SAWSDL location),
- candidateOps, a list of Web service operations to consider for suggestions (Operation name and WSDL/SAWSDL location) and
- desired functionality specified as a concept in the Ontology or as keywords

The SSE Web service returns a ranked list of Web service operations along with the matching scores in XML or JSON format.

Galaxy is a Web-based platform to facilitate data intensive biomedical research and the construction and execution of bioinformatics and biomedical workflows. Galaxy provides an
easy way to construct workflows from tools that are built into Galaxy. Through our work [4] we have developed a tool that allows adding Web services as tools to Galaxy. This addition has made Web service composition possible in Galaxy. In order to facilitate invocation of SSE (as a RESTful Web service) to provide suggestions to the user for the next or previous steps, we have extended Galaxy’s workflow editor interface (figure 2.5). The user can, if they desire, use SSE during the construction of a workflow. The interface also allows the user to optionally specify the desired functionality for the next or previous step.

Taverna is an open source Workflow Management System for design and execution of scientific workflows, that is available as a desktop application. Taverna is developed to be modular in nature, which facilitates development of plugins to extend or complement the
core functionality. Similar to the extensions to Galaxy, a plugin can be created for Taverna to provide suggestions in the workflow construction process by invoking SSE.

2.5 Evaluation

For the purposes of evaluation, we have considered a common, real-world scenario often encountered by bioinformaticians, that of finding more information about a protein sequence and its evolutionary relationships to other protein sequences. The user who wants to create a workflow for this scenario might have some basic idea, that he or she wants to first search a database for similar sequences, then do multiple sequence alignment of the sequences returned by the sequence similarity search and finally perform phylogenetic analysis to construct a phylogenetic tree to discover the nature of the relationships [11].

Though the workflow from a user’s point of view performs three basic data analysis operations, creating a Web service composition for the same requires more steps than a user, accustomed to performing the analysis via a browser and Web-based servers, would anticipate. Typically, for bioinformatics Web services, each Web service execution is comprised of two steps, one for the actual execution and other for retrieving the results of the execution. Also in some of the cases, additional Web service operations might be required in order to convert the data into a required format by the Web service operation. The need for these additional steps make the creation of Web service compositions more difficult for the novice user. A complete workflow for the above scenario is comprised of nine steps and primarily uses WU-BLAST for searching similar sequences, ClustalW2 for multiple sequence alignment and wsPhylipProtDist and wsPhylipNeighbor for construction of phylogenetic trees. We have evaluated SSE for forward, backward and bidirectional suggestions against a consensus ranking by human experts for each of the cases. For forward suggestions, we consider that the first Web service operation has been added to the workflow by the user and evaluate
<table>
<thead>
<tr>
<th>Steps</th>
<th>Web Service and Operation Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>WU-BLAST.run</td>
<td>Sequence Similarity Search</td>
</tr>
<tr>
<td>Step 2</td>
<td>WU-BLAST.getResult</td>
<td>Sequence Similarity Search</td>
</tr>
<tr>
<td>Step 3</td>
<td>wsFilterSequences. filterByEvalScore</td>
<td>Utility Web Service</td>
</tr>
<tr>
<td>Step 4</td>
<td>ClustalW2.run</td>
<td>Multiple Sequence Alignment</td>
</tr>
<tr>
<td>Step 5</td>
<td>ClustalW2.getResult</td>
<td>Multiple Sequence Alignment</td>
</tr>
<tr>
<td>Step 6</td>
<td>wsPhylipProtDist.protdistDefaultParameters</td>
<td>Phylogenetic Analysis</td>
</tr>
<tr>
<td>Step 7</td>
<td>wsPhylipProtDist.retrieveProtDistResult</td>
<td>Phylogenetic Analysis</td>
</tr>
<tr>
<td>Step 8</td>
<td>wsPhylipNeighbor.runNeighbor</td>
<td>Phylogenetic Analysis</td>
</tr>
<tr>
<td>Step 9</td>
<td>wsPhylipNeighbor.retrieveNeighborResult</td>
<td>Phylogenetic Analysis</td>
</tr>
</tbody>
</table>

Table 2.2: Steps in the Workflow

the performance of the suggestions made for the next eight steps. For backward suggestions, we assume that the last operation is added to the workflow by the user and evaluate the performance of the suggestions made for the first eight steps. Along similar lines, bidirectional suggestions are considered for seven middle steps, step 2 to step 8. Table 2.2 shows the Web service operation for each of the nine steps in the workflow.

The SSE is used to suggest Web service operations from a candidate list of 101 semantically annotated operations from 19 different Web services belonging to different categories like Sequence similarity search, Multiple sequence alignment, Protein functional analysis, Phylogenetic analysis as well as some utility Web services for functions like format conversion. As discussed previously, all the Web services we are utilizing have been annotated with the OBI-WS Release version. A complete list of annotated SAWSDLs and the version of the ontology used for the evaluation can be found at http://mango.ctegd.uga.edu/jkissingLab/SWS/services.html. When constructing a workflow there are often multiple possibilities for a user to consider when selecting the next or previous steps depending upon what the user intends to do. For example, if the user has some protein sequences from a previous step and wants to perform multiple sequence alignment, she can do so with operations from Web services like ClustalW2, T-Coffee or MUSCLE, all of which perform multiple
sequence alignment. SSE categorizes the operations into two groups (1) High: those operations that are highly suitable for use in the next step and (2) Low: those which are not suitable to be used for the next step. The results of our human consensus provide a list of highly probable operations for the next step. We use precision and recall as measures for the quality [32] of suggestions made by the SSE against the human consensus. Precision ($P$) is the fraction of retrieved results that are relevant and is calculated as,

$$P = \frac{\text{RelevantResults} \cap \text{RetrievedResults}}{\text{RetrievedResults}}$$  \hspace{1cm} (2.10)

Recall ($R$) is the fraction of relevant results that are retrieved and is calculated as,

$$R = \frac{\text{RelevantResults} \cap \text{RetrievedResults}}{\text{RelevantResults}}$$  \hspace{1cm} (2.11)

The F-measure is the harmonic mean of precision and recall. An important aspect of SSE is narrowing down possible options for next or previous step in the workflow by considering input-output and functionality matching. Hence, for this study, we are using a variation of F-measure $F_\beta$ with $\beta = 2$ which weights recall higher than precision. $F_\beta$ is calculated as

$$F_\beta = (1 + \beta^2) \frac{P \ast R}{\beta^2 \ast P + R}$$ \hspace{1cm} (2.12)

The graphs in figure 2.6 summarizes the $F_2$-measure for all the steps in the workflow for forward suggestions when using path-based and p-homomorphism data-matching algorithms. The different lines, distinguished by the legend in the graph stand for different levels of annotation. The steps here correspond to the steps listed in table 2.2. As was expected, use of unannotated Web services did not yield any useful results. When suggestions are made from unannotated Web services, different string metrics like the Levenshtein’s distance and the Dice algorithm are used to compute the similarity between two WSDL elements.
which resulted in inaccurate results due the lack of any standard naming convention (figure 2.6). The $F_2$-measure for Web services with semantic annotations, without any desired functionality supplied by the user averages to 0.65 (figure 2.6). When desired functionality is provided in addition to semantic annotations on inputs and outputs, the results are expected to be better. However, this was not always the case. While the $F_2$-measure when desired functionality was provided (as keywords) increased for some of cases (4 out of 8) it dropped in others (see figure 2.6).

![Figure 2.6: Comparison between Forward Suggestions for Path-Based and p-Homomorphism Input-Output Matching Algorithms at Different Levels of Annotation](image)

The user is assumed to be unaware of the fact that most Web service execution involves two steps, one for actual execution and other for retrieving the results. Hence, the user might provide a functionality describing what she wants to do next rather than the necessary function of retrieving the results. This causes the relevant operations to be actually weighted lower and can result in a lower $F_2$-measure as in steps 2 and 6, in spite of this, the average $F_2$-measure of 0.67 showed a marginal increase. SSE also allows the user to specify the desired functionality as a concept in the ontology and this case resulted in a marginally
higher $F_2$-measure of 0.69.

The right side of figure 2.6 shows the $F_2$-measure for forward suggestions when using the pHomomorphism matching algorithm for calculation of the input-output matching scores. The pHomomorphism is an approximate sub-graph homomorphism-mapping algorithm that considers the entire structure of the XML inputs and outputs unlike path-based data matching which decomposes the input-output tree structure into paths. This offers a somewhat stricter matching algorithm, which is evident from the results in figure 2.6. Both step 2 and step 5 where the algorithm performs worse are the `getResult` steps for WU-BLAST and ClustalW2, respectively. There are different steps that can be performed after the execution step, like retrieving the status of execution `getStatus` or retrieving the supported result types for the output `getResultTypes`. pHomomorphism favors these over the `getResult` (the preferred choice) by a heavy margin. The calculation of functionality sub-score works in the same way to path-based data-matching. The average $F_2$-measure for pHomomorphism matching (without functionality) is 0.62, a little lower than the respective $F_2$-measure for path-based input-output matching, while the average $F_2$ measures when functionality was provided as keywords and as an ontology concept are 0.60 and 0.61, respectively.

The graphs in figure 2.7 plots $F_2$ measures for path-based and p-homomorphism input-output matching algorithms for forward, backward and bidirectional suggestions. Except for the two `getResult` steps mentioned before, p-homomorphism actually performs as good or better than the path-based algorithm, but still gives a lower average $F_2$-measure than the path-based algorithm.

When going in the backward direction, both the algorithms perform quite well with path-based and p-homomorphism data-matching algorithms yielding an average $F_2$-measure of 0.77 and 0.85, respectively. In some of the steps, backward suggestion eliminates some of the possible choices which works in the favour of the algorithms, as visible in figure 2.7.

Each bidirectional suggestion performs two instances of input-output matching, one with
the inputs of the previous operation and another with the outputs of the next operation. The performance of bidirectional suggestions, figure 2.7, is somewhat mid-way between forward suggestions and backward suggestions with an $F_2$-measure of 0.7 and 0.68 for path-based and p-homomorphism data-matching algorithms, respectively. From the evaluations, the path-based data-matching algorithm is a safer choice to go with.

Figure 2.7: Comparison Between Path-Based and pHomomorphism Input-Output Matching Algorithms for Forward, Backward and Bidirectional Suggestions
2.6 Related Work

In recent years, work has been done to advance the area of service composition, especially Web service composition (WSC) using Semantic Web Services. This has become extremely important in the composition process for scientific workflows as more and more workflow components have been adapted into Web services with varying degrees of complexity in their inputs and outputs.

The work in [33], outlines the general issues in workflow composition and the need for assistance in workflow compositions. It discusses the benefits of describing a workflow using application-specific ontology in order to facilitate semi-automatic workflow composition and service discovery.

In 2010, Withers et al. [34] compared and contrasted two Semantic Web Service frameworks that have plugins for Taverna: BioMoby and the Semantic Automated Discovery and Integration (SADI) Framework. The BioMoby project developed with an aim of facilitating and standardizing interoperable information exchange in bioinformatics, utilized an ontology-based XML schema in order to semantically represent complex, but predictable, XML structures and substructures. However, this approach meant that all Web services that were applicable for discovery by the system needed to be BioMoby compliant. The SADI Framework utilizes both RDF and OWL and takes service inputs and outputs as OWL-DL classes such that individuals of these classes are consumed and produced when the service is invoked. In order to facilitate discovery, applicable services needed their inputs and outputs to be decorated with RDF relationships amongst themselves. The downside to this approach is also one of compliance. The relationship between the input and output of the service needed to be clearly defined in the ontology, and each service operation required its own OWL-DL class definition.

In 2011, Zhang et al. [35] researched the area of semantically-empowered biomedical
service composition that makes use of semantically annotated WSDL files (i.e., SAWSDL files) in order to facilitate the semantic relationships between input and outputs of a service. The approach described utilizes a basic ontology that describes both the static and behavioral semantics of generic services and attempts to automatically annotate WSDL files with concepts from this ontology. Much information is provided about the architecture of the overall system, however, no statistical comparisons to workflow compositions created by domain experts were provided, just as with previously described efforts.

Focused on workflow composition in the biomedical imaging domain, the research presented in Wali et al. [36] describes the use of OWL-S, an OWL ontology for describing the functionality and execution of Web services, as well as extensions to OWL-S to aid in adapting legacy Web services so that they can be used by workflow composition systems. A comparison between WSDL-S, SASWDL and OWL-S is provided, and, although the authors clearly state the decoupling benefit of SAWSDL, they give no compelling argument for why SAWSDL cannot be used in conjunction with the OWL-S ontology.

### 2.7 Conclusions and Future Work

Semantic Web Services envision a Web in which Web services are described using rich semantics which will enable the automation of Web service discovery and composition. Markup of Web services to make them computer interpretable and agent ready is referred to as a fundamental component of the Semantic Web. The success of applications that make use of semantically annotated Web services to realize Web service compositions and discovery largely depend on the availability of semantically annotated Web services and ontologies to describe them. The development of ontologies and tools to speed up the process of semantic annotation of Web services in turn depend on the tools that could put them to use. This is commonly referred to as the Semantic Web chicken and egg problem [37].
The use case demonstrated here in our workflow, that of choosing from 101 operations for creating a 9-step workflow, presents a challenging task for a user if it has to be performed without tool support. The problems associated with Web service compositions like finding Web service operations with desired functionality and that are input-output compatible will only get worse with the ever-increasing number of Web services and availability of data in disparate data formats. With an average $F_2$-measure of 0.7, SSE can help the user considerably narrow down the choices for the next or previous step. The performance of SSE also provides a way for assessing the quality of semantics developed through OBI-WS.

The availability of a Service Suggestion Engine as a Web service will make it easier to integrate it with existing workflow composition tools. The tool that is developed to add Web services to Galaxy, currently does not support complex output types, primarily due to the way Galaxy handles the output of tools in the workflow. The workflow considered in the evaluation can be executed in Galaxy if the tool for adding Web services to Galaxy handles complex output types as discussed in appendix A. However, an alternative workflow, that uses \texttt{WsDBFetch} instead of \texttt{wsFilterSequences} is available on the Web site and can be currently executed in Galaxy. The current workflow that uses XML output for WU-BLAST and \texttt{wsFilterSequences} Web service was chosen in order to illustrate the capabilities of input-output matching algorithms. Also, we want to evaluate SSE with additional set of workflows, Web services and test users. We are planning to create a plug-in for Taverna along the lines of the Galaxy extensions that would allow invocation of SSE from Taverna to provide help in workflow creation. Every Galaxy tool is accompanied by a tool-configuration file that serves the same purpose as a Web service description document. We are planning to annotate the Galaxy tool-configuration files so that the SSE can provide suggestions for Galaxy tools as well. The schema mapping attributes proposed by SAWSDL allow for specification of a mapping between XML instance data and semantic data or vice versa. We plan to use lifting and lowering Schema Mappings in a future version of SSE. The original version of SSE

34
considered pre-conditions and effects, but due to their complexity were removed from this version. A later version supporting local planning would require restoration of pre-conditions and effects.
Chapter 3

SUMMARY

The Service Suggestion Engine uses semantically annotated WSDLs (SAWSDLs) to make suggestions for the next, previous or intermediate step by considering how well the inputs and outputs match and how well the objective specification of the operation matches with the user-specified desired functionality. The algorithms implemented by SSE, path-based data-matching, p-homomorphism mapping and concept similarity are not specific to a domain and we believe that SSE can be easily adapted to provide suggestions for workflow creation in other domains as well. The evaluations of the Service Suggestion Engine attest to the usefulness of the system in providing help for workflow creation. As discussed earlier, choosing from 100 Web service operations for a workflow, by looking at the XML WSDL files, is an unrealistic task for a user. SSE can help the user narrow down the choices for the next/previous/intermediate step. The performance of SSE also provides a way for assessing the quality of semantics developed through OBI-WS. Every possible effort was made while modeling Web services to ensure the correctness of the knowledge represented. Creating the necessary semantics is definitely a time-consuming process. But our experience has shown that the effort expended reduces considerably as more and more Web services in the domain are modeled. Additionally, the availability of tools like the RadiantWeb annotation tool
makes the semantic annotation process much simpler.
REFERENCES


[18] Steve Pettifer, Jon Ison, Mat Kala, Dave Thorne, Philip McDermott, Inge Jonassen, Ali Liaquat, Jos M. Fernndez, Jose M. Rodriguez, INB Partners, David G. Pisano,


[34] David Withers, Edward Kawas, Luke McCarthy, Benjamin Vandervalk, and Mark Wilkinson. Semantically-guided workflow construction in Taverna: the SADI and


APPENDICES

1. Appendix A, is a developers guide for Service Suggestion Engine. That describes ways to invoke, use or extend SSE.

2. Appendix B, provides for comparison between two popular Semantic Web Service Frameworks, SAWSDL (used in this project) and OWL-S which is an upper level ontology for Web services.
Appendix A

Developers Guide

The source code and binaries for SSE (*Current release v0.9*) is available MIT license and can be downloaded from the project Web page\(^1\). The earlier versions of SSE can be downloaded from the GitHub Repository by cloning the the GitHub repository using the command `git:repositoryName`\(^2\). The suggestions capabilities of SSE have been exposed through a RESTful API and can be accessed from its endpoint\(^3\). The JavaDocs for SSE can also be accessed from the project page\(^4\). The Package OntologySimilarity that provides functionality to calculate a similarity score between two concepts in the same Ontology can be downloaded from here\(^5\). It considers both syntactic and logical definition of an OWL Class. It takes into account (1) relative Position of two concepts in the Ontology (Coverage Score), (2) syntactic definition (Definition + Label: Syntactic Score) and properties and restrictions on them for the OWL class in consideration The main Package OntologySimilarity has two Internal dependencies: OntologyManger (For Loading and processing the Ontology), and StringMetrics (Source code for Both added to the Distribution ). Ontology Manager uses

\^1Download Source Code: http://mango.ctegd.uga.edu/jkissingLab/SWS/sse.html
\^2GitHub Repository: https://github.com/mepcotterell/SuggestionEngine
\^3http://wsannotations.ctegd.uga.edu/SSE/
\^4JavaDocs: http://mango.ctegd.uga.edu/jkissingLab/SWS/sse/javadoc/
\^5Ontology Similarity: http://mango.ctegd.uga.edu/jkissingLab/SWS/utilities.html
OWL API to load the Ontology, its imports, and retrieve various aspects of an OWL class required for computation of the Similarity.
Appendix B

SEMANTIC WEB SERVICE FRAMEWORKS: SAWSDL VS. OWL-S

B.1 Introduction

Semantic Web envisions a Web in which the data are described using rich semantics that allow more effective exchange of data between various applications and enable answering complex queries [38]. The work in [39] refers to markup of Web services to make them computer interpretable and agent ready as a fundamental component of Semantic Web. The term Semantic Web service has been used and talked about in the literature since early 2000s. The article Semantic Web Services [39] was one of the seminal works in this field that proposed a framework for semantic Web services (SWS) and proposed that DAML be used for semantically modeling Web services. Some of the Semantic Web Service frameworks include WSDL-S (SAWSDL), OWL-S and Web Service Modelling Ontology. In this essay we will go over and compare the features offered by two of the popular semantic frameworks
SAWSDL and OWL-S.

B.2 OWL-S: Semantic Markup for Web Services

OWL-S [40] is an ontology for describing Web services developed with the aim of realizing automatic (or semi-automatic) Web service discovery, invocation and composition as envisioned for Semantic Web Services (SWS). OWL-S models Web services by organizing the service description into three top level classes or concepts: Service Profile, Service Model and Service Grounding (see figure B.1).

The service class itself is an abstraction of a Web service such that each Web service will be represented by an instance of the service class. Service Profile models all the information that would be necessary to describe what the Web service does for the purpose of advertising the Web service for Web service discovery. Service profile models this information in terms of the Web services parameters (inputs and outputs), preconditions and results, service category, service classification and more. The Service Model is a more detailed model of Web services operation (as a process) providing enough information to interact with the Web service. The Service Grounding specifies how the service can be accessed in terms of the communication protocol, message formats and port numbers.

B.3 Semantic Annotations for WSDL and XML Schema (SAWSDL)

Semantic Annotations for WSDL and XML Schema (SAWSDL) [5] is a W3C Recommendation that provides a simple mechanism for adding semantics to Web services in bottom-up manner. SAWSDL defines three extension attributes for this purpose that could be used to annotate different aspects of a WSDL document like operations and input-output messages.
The extension attributes provide support for WSDL 1.1, WSDL 2.0 and XML schema definitions. SAWSDL defines following three extension attributes:

**modelReference:** The modelReference is used to annotate WSDL and XSD elements with concepts/classes from a semantic model, typically an Ontology. The Semantic annotations are references (URIs) that point to the relevant ontology concept to be annotated with. Model reference can be used to annotate both simple and complex types. Complex types can be annotated by adding either bottom level or top level annotations as per the needs of the application.

The schema matching attributes have been introduced to specifically address the issues concerning heterogeneities between with XML input-output messages. The schema matching attributes allows specifying matching between the semantic model and XML data (see figure B.2). SAWSDL does not specify any language for matching, but some of the languages
that can be used are XSLT, SPARQL or XQuery. The schema-matching attribute points to an XSLT/XQuery document that defines the transformation from XML to semantic data or vice versa.

**liftingSchemaMapping:** The liftingSchemaMapping attribute specifies the transformation of XML instance data to semantic data.

**loweringSchemaMapping:** The loweringSchemaMapping attribute specifies the transformation of semantic data to XML instance data.

### B.4 Observations

#### B.4.1 The OWL-S Top-down and SAWSDL Bottom-up Approach

The top-down approach that OWL-S follows, assumes that the Web service developer first models the Web service semantics in terms of Service Profile and Service Model and then provides the service implementation [41]. This OWL-S top-down approach makes it difficult to model the large number of existing Web services.

SAWSDL, by providing the extension attributes for adding semantics to Web services, makes it possible to provide semantic annotations in a bottom up fashion. This makes it easy to semantically model the large number of existing Web services. The SAWSDL specification does not specify a semantic model or ontology language to use. That is WSDL or XSD elements can be annotated with concepts from OWL, RDFS or OBO ontologies alike. OWL-S on the other hand is purely Web Ontology Language (OWL) based.

#### B.4.2 Support for Data Mediation in Web service Compositions

OWL-S specification, through both Service Profile and Service Model provides support for specifying preconditions and effects (results), but there is no explicit support for handling the
input-output heterogeneities [42]. Preconditions specify conditions under which the desired output can be expected, while effects specify the change in state produced once the operation is executed and they both can play an important role in automating service compositions. Though there is no explicit provision for handling heterogeneities in OWL-S, the modeler can specify the translation in service grounding.

The schema matching attributes proposed by SAWSDL allows for specifying the matching between the XML instance data and semantic data or vice versa. The semantic model serves as an intermediate link between the heterogeneous inputs and outputs (Figure B.2). For example the lifting schema mapping transformation can be used to transform the XML output to semantic data, while at the other end the lowering mapping can be used to transform this semantic data into instance data, which now can be fed into the next Web Services input.

The work in the paper [43] proposes a system architecture that utilizes the extension attributes supported by SAWSDL. The data mediation component handler can be built into
A Web service engine such as Apache Axis2 \(^1\) or it can be offered as a Web service itself as suggested in [44].

**B.4.3 Practical Issues**

OWL-S is basically an ontology for modeling Web services and the descriptions or specifications of all the Web services modeled would be added as individuals to the respective upper level terms defined by OWL-S for this purpose, e.g., inputs, outputs, preconditions, results, etc. This makes it necessary that the person who models the Web services (irrespective of top-down or bottom-up approach) is an ontologist, who also has the necessary domain knowledge. Due to increasing number of Web services and limited reuse of terms in OWL-S the size and number of ontologies for describing Web services might increase considerably over time, especially with respect to ontologies that would be required when SAWSDL. The top-down approach puts the burden of providing semantics on the Web service developers. A tool, OWL-S Editor [45], has been developed as a plugin for Protege to ease the process of developing OWL-S services.

In case of SAWSDL, once the required vocabulary for describing the Web services in a particular domain is created in the form of a domain ontology, annotation of Web services could be easily done by an annotator who has some amount of domain knowledge. In addition to that, availability of a tool like RadiantWeb [21] for automating the annotation process makes it even easier to semantically annotate Web services with SAWSDL.

\(^1\)Axis2 http://axis.apache.org/axis2/java/core/