RADIANTWEB: A TOOL FACILITATING SEMANTIC ANNOTATION OF WEB SERVICES

by

CHAITANYA GUTTULA

(Under the Direction of John A. Miller)

ABSTRACT

Semantic Web services are playing an important role in the Web 2.0 and the Semantic Web. Semantic Web services provide several advantages for Web service discovery, composition and invocation. Research in the area of annotation mechanisms led to the formation of specifications like SAWSDL, WSDL-S and SA-REST. My work focuses on tool to support the annotation of Web services (both SOAP and REST services) in both a manual and semi-automated ways. The tool greatly aids users in annotation of Web services. I achieved the matching of the Web service description elements to ontological concepts with the help of different string similarity algorithms. My matching considers the definitions of the concept in the ontology and the documentation of the elements in the description documents for getting the recommendations. An evaluated accuracy of the recommendations provided by the tool while annotating Web services from the European Bioinformatics Institute indicates the utility of recommendations.

INDEX WORDS: Semantic Web, Web Services, SAWSDL, WSDL-S, SA-REST, Web service discovery, SOAP, REST, Web 2.0, ontology
RADIANTWEB: A TOOL FACILITATING SEMANTIC ANNOTATION OF WEB SERVICES

by

CHAITANYA GUTTULA

B.Tech., Andhra University, India, 2008

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
RADIANTWEB: A TOOL FACILITATING SEMANTIC ANNOTATION OF WEB SERVICES

by

CHAITANYA GUTTULA

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

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2012
DEDICATION

To my family and friends.
ACKNOWLEDGEMENTS

I am greatly thankful to my major professor, Dr. Miller for his continuous support and guidance during my study at UGA. He is a wonderful professor and always encourages his students. I would also like to thank Dr. Kissinger for her valuable suggestions and support. I am thankful to Dr. Kochut, who has introduced me to Semantic Web and helped in understanding the concepts in my first semester.

Finally, I would like to thank my colleagues who have helped me on this project: Alok Dhamanaskar, Jie Zheng, Yung Long Li, Akshay Vivek Choche and Micheal A. Cotterell.
TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................................................... v

LIST OF TABLES ......................................................................................................................... viii

LIST OF FIGURES ..................................................................................................................... ix

CHAPTER

1 INTRODUCTION ......................................................................................................................... 1

2 BACKGROUND ......................................................................................................................... 4
   2.1. Semantic Web Services ................................................................................................. 4
   2.2. Semantic Annotations .................................................................................................. 6
   2.3. Motivation ..................................................................................................................... 9

3 TECHNIQUES FOR ANNOTATING WEB SERVICES ........................................................... 11
   3.1. Editing WSDL/WADL Documents .............................................................................. 11
   3.2. Using GUI Based Annotation Tool ............................................................................ 12
   3.3. Automation – Providing Recommended Annotations ................................................. 12
   3.4. Recommendation Mechanisms .................................................................................. 12
   3.5. RadiantWeb Annotation Tool .................................................................................... 17

4 SYSTEM ARCHITECTURE ....................................................................................................... 20
   4.1. Components ................................................................................................................ 21
   4.2. Usage .......................................................................................................................... 29

5 RELATED WORK ..................................................................................................................... 31
6 EVALUATION .................................................................................................................. 33

7 CONCLUSIONS AND FUTURE WORK ........................................................................ 36

REFERENCES ...................................................................................................................... 38

APPENDICES

A USER GUIDE .................................................................................................................. 43

B SAMPLE ANNOTATED DOCUMENT ............................................................................. 49

C JAVA DOC OF THE PACKAGES .................................................................................. 50
LIST OF TABLES

Table 1: Conditions for weighing different similarity score components............................ 15
Table 2: Accuracy of Inputs outputs and operations for both the cases.................................... 35
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RadiantWeb annotation tool</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>System Architecture</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Comparison of the document viewer of RadiantWeb in different versions</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Model Component of the System</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Code snippet to get the leaf nodes of a given operation</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>Code snippet to calculate the individual match scores</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Comparing the accuracy of recommended concepts in both cases</td>
<td>34</td>
</tr>
<tr>
<td>8</td>
<td>Graph showing the accuracy of recommendations for each service in both cases</td>
<td>34</td>
</tr>
<tr>
<td>9</td>
<td>Graph showing accuracy for inputs, outputs and operations in both cases</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>Description document loader</td>
<td>43</td>
</tr>
<tr>
<td>11</td>
<td>Ontology loader</td>
<td>44</td>
</tr>
<tr>
<td>12</td>
<td>Recommendations provided by the tool for the entire document</td>
<td>45</td>
</tr>
<tr>
<td>13</td>
<td>top-k recommendations for a particular part of the document</td>
<td>46</td>
</tr>
<tr>
<td>14</td>
<td>Ontology search</td>
<td>46</td>
</tr>
<tr>
<td>15</td>
<td>Ontology search results</td>
<td>47</td>
</tr>
<tr>
<td>16</td>
<td>Fragment of WUBLAST.sawSDL</td>
<td>49</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Web services are applications that have well defined interfaces (REST or SOAP) that can be programmatically invoked by other software or tools over the Web [1]. Considering the Web-accessible nature and flexibility that Web services provide, the number of tools and software applications available as Web services has increased dramatically in recent years. For example, from May 2010 to April 2012, BioCatalogue [2], a curated registry of Life Science Web Services, increased the number of biomedical-related Web services it lists from 1627 to close to 2290. Similarly, seekda [3], a general purpose Web service repository, discovered more than 28,000 Web services over a 68-month period. Such increases present an interesting challenge to users wishing to find the right Web service for a particular job.

Ongoing research in Semantic Web services envisions some degree of automation in the discovery, composition [4, 5] and invocation of Web services by making use of semantic annotations. Semantic annotations are references that connect various elements of a Web service, described in a description document, with concepts from a semantic model or ontology. Patil et al. [6] discuss about how semantic annotations from an ontology can ease the problem of finding the right Web service.

There is a need to semantically annotate the large number of available Web services. Each Web service is typically described by a Web service description document in terms of its operations and inputs and outputs. Semantically annotating Web services mainly requires finding an appropriate ontology and searching for relevant concepts to annotate different aspects of the
description document, e.g., operations, inputs and outputs. A single Web service usually supports multiple operations, each with several inputs and outputs. This makes it more difficult to annotate the thousands of available Web services. An annotation tool that provides some degree of automation can substantially reduce the effort required.

The work done by Patil et al. [6] for providing semi-automated annotation assumed that the names of the elements in the description document and ontology are meaningful. However, there is no standard naming convention followed in naming the elements in the description documents. Ontology developers model the labels for the concepts at more abstract level, whereas the Web service programmers name the element at more specific level, in which case matching approaches based upon names will fail. Improved matching will result if labels as well as definitions are utilized. In addition, similarity matches between the labels/definitions of concepts in the ontology and the names of elements in the Web service description document may not suffice. In many cases, the elements of the WSDL/WADL document are accompanied by documentation, which can provide additional information about the element. We make use of this information provided by the description documents (WSDL/WADL).

The outcome of this thesis is a Web-based tool called RadiantWeb to facilitate semi-automatic semantic annotation of Web services. The tool provides several features including ontology search, display of documentation of elements, display of the definition of a selected concept, and recommending concepts for annotation. The tool loads the description document by trimming its complex structure and displaying the operations and its inputs and outputs as a tree structure making it easy for users to use the tool. The tool also provides the option of viewing the eXtensible Markup Language (XML) form of the documents for advanced users. Support for the searching concepts in the ontology given a set of keywords considerably eases the ontology look-
up process, as the sizes of real-world ontologies are typically huge. After the system recommends the concepts, the user can verify them with the help of the definitions of the recommended concept and the documentation of the description document elements.

The rest of the thesis is organized as follows. Chapter 2 provides the background on Web services and semantic annotations for Web services. Different techniques that can be used for annotating Web services are explained in chapter 3. Chapter 4 explains the architecture of the tool and different components involved in it. Chapter 5 discusses the related work. Chapters 6 and 7 cover the evaluations and conclusions, respectively.
CHAPTER 2

BACKGROUND

Web services can be described using the Web Service Description Language [8] (WSDL) or Web Application Description Language [9] (WADL). WSDL and WADL are XML based specifications that formalize ways to describe Web services. These description documents include information about the operations supported and their inputs and outputs. This information can be used by tools for invocation, e.g., Apache Axis2 [10], and discovery of Web services, e.g., seekda.

A WSDL document has two parts: one an abstract definition and the other a concrete network deployment specification. Types, messages, operations and port types form the abstract part, while bindings, ports and services form the concrete part of the WSDL, which mainly provide details for invocation of the service. The two versions of WSDL commonly used are WSDL 1.1 and WSDL 2.0. WSDL 1.1 can only describe SOAP Web services, whereas WSDL 2.0 can describe both SOAP & REST [11] Web services. The WADL specification describes the REST Web services, which are generally HTTP-based Web applications, in terms of resources associated with the methods/operations. Each method has inputs and outputs represented as request and response, respectively. It also gives information on how to construct a URL for invoking the service.

2.1. Semantic Web Services

The information provided by the description document in terms of the operations contained in a Web service, as well as their inputs and outputs, has some semantic
information such as the type of the element, it is inherently ambiguous on what data it can take. For example, `<element name="studentid" type="string">` the element studentid allows only string values but the value can be any value that is a string. Hence, though this information can be used in Web service composition and discovery, it often requires human intervention. Additional meta-data, in the form of semantics that is machine interpretable and unambiguous can enhance the level of automation in the areas of Web service discovery, composition and invocation. A semantic model like an ontology provides a rich modeling framework to capture real-world entities and can act as a knowledge base. Ontologies tend to focus on nouns (things in the real world) and can also be used to model actions or processes. The nouns are useful for describing the inputs and outputs of Web service operations, while the actions/processes are useful for describing functionality. Also, ontologies facilitate community agreement, reasoning to ensure consistency and enable reuse of domain knowledge. These features make it suitable for the semantic annotation of Web services. As experimental data is tremendously increasing in the field of biology, storing this data in the ontologies would facilitate better reuse, rather than storing in traditional databases. The semantic Web services can make use of these ontologies to provide interoperability between different bioinformatics tools.

Different semantic frameworks, such as OWL-S [12], Web Service Modeling Ontology (WSMO) [13], WSDL-S [14], have been developed to by the Semantic Web Services (SWS) community. OWL-S and WSMO follows a top down approach (i.e., the semantics should be kept in mind while designing the Web services) making it hard for them to comply with the present standards like WSDL and REST [15]. OWL-S is an upper level ontology to model the Web services and mainly consists of three interlinked sub-ontologies: service profile, process model, and service grounding. The service profile describes the functionality of a Web service in terms
of its inputs, outputs, pre-conditions and effects (IOPE) and is mainly used for discovery purposes. A process model describes how the service is used and is utilized during the invocation of the service. Finally the service groundings give the concrete details such as the message format, transport protocol and addressing. The groundings can be mapped to the concrete network part of a WSDL whereas the service profile and process model map to the abstract part of the WSDL [12]. "WSMO is a top-down conceptual model for SWS that defines four top-level components: ontologies, mediators, goals and Web services" [15]. As WSMO and OWL-S are heavy weight frameworks, WSDL-S has been developed to reference ontology concepts in a lightweight fashion, the basis for Semantic Annotations for WSDL (SAWSDL).

SAWSDL [16], a World Wide Web Consortium (W3C) [1] recommendation, is an extension to WSDL to support semantic annotations in a bottom up fashion. SAWSDL defines a lightweight mechanism to add semantics to existing WSDL documents using extension attributes. The model reference attribute can be used to refer to a concept in an ontology, while schema-mapping extension attributes allow the specification of lifting or lowering schema mappings (as transformations) between the WSDL elements and ontological concepts.

2.2. Semantic Annotations

Semantic annotations are the links that reference the WSDL/WADL element with concepts from a semantic model or ontology. The semantic annotations for Web services can be classified as data semantics, functional semantic, execution semantics and Quality of Service (QoS) semantics in a Semantic Web process lifecycle [17]. The data semantics and functional semantics are the annotations on the input/output and operation, respectively, which are used during the discovery and interoperation of Web services [17]. Our work mainly focuses on data and functional semantics.
In a study conducted by Wang et al. [4], it was found that the effectiveness of semantic annotations comes primarily from functionality annotations, secondarily from annotations on inputs and outputs, and thirdly from precondition/effects specifications. The first two types of annotations are much simpler to provide than the third. The RadiantWeb annotation tool currently provides support for functionality annotation and annotation on inputs and outputs.

2.2.1. Semantic Annotation for WSDL

W3C has recommended a specification, Semantic Annotation for WSDL, which defined a set of extension attributes for the Web Services Description Language and XML Schema Definition (XSD) language, to add semantic annotations to various parts of a WSDL document such as input and output message structures, interfaces and operations. The specification provides two mechanisms for annotation, model reference and lifting & lowering schema mappings.

- **Model reference** is an extension attribute to specify the association between a WSDL or XML schema component and a concept in some semantic model (e.g., ontology). Model references are usually attached to the WSDL elements such as operation, fault and port type and to XML schema components such as element, simple type and complex type. A model reference has the capability of taking multiple URIs from different ontologies. Ideally, these ontologies should be aligned in order to make more effective use of the annotations.

- **Lifting and lowering schema mappings** are two extension attributes that are added to XML schema element declarations and type definitions for specifying mappings/transformations between semantic data and XML data. These transformations
are generally specified using the eXtensible Stylesheet Language Transformations (XSLT), XQuery or SPARQL Protocol and RDF Query Language (SPARQL). For example, consider a WSDL having two fields for name as first name, last name and middle name and an ontology has a concept "full name". In this case, schema mappings are used to settle the heterogeneities between both the formats.

Annotating simple types is trivial, but annotating complex types is not straightforward and is achieved in two ways: bottom level annotation and top-level annotation. In bottom level annotation, annotations are added to the leaf nodes in the complex structure. The top-level annotation is targeted at annotating the root element of the complex structure.

2.2.2. Semantic Annotations for REST Services

The aforementioned SAWSDL specification works only for WSDL documents. At present, very few REST Web services are described using WSDL and WADL, but most are described in an ad-hoc manner in unstructured HTML/Text. There is a need for an annotation mechanism for the REST services that are not described in WSDL. SA-REST [18] has been developed for annotating REST Web service descriptions that are present in the Web in an unstructured way. In our work, extension attributes are used to annotate WADL documents, analogously to how WSDL documents are annotated.

As there is no standard specification recommended for annotating WADL documents, we have prepared a specification for annotating WADL documents. It uses the same extension attributes, model references and schema mappings, employed by SAWSDL, just on different elements. A model reference can be used to annotate different parts of a WADL document, such as an application, resource, method or XML element (e.g., simple type, complex type, element).
A schema mapping annotations can be added only to the XML components such as complex types, simple types and element. For more details see [19].

2.3. Motivation

The purpose of annotations is to provide formalized documentation that can be read by humans and processed by machines. Annotating via model references offers multiple benefits, including (i) reuse of documentation, (ii) reduction of ambiguity and (iii) formalization of the means for describing similar concepts and how they relate to each other. All this can give the algorithms/programs a better understanding of the Web service in general with respect to what it does, what inputs it takes and outputs it produces. The annotated description documents can facilitate Web service discovery and Web service workflow compositions.

In order to reap the benefits provided by semantic annotations, a large number of Web services need to be annotated. Considering the human effort required for annotation, this is unlikely to be put into practice without appropriate tool support. Manual annotation is expensive, as it requires several hours of efforts by a human annotator who has the required domain knowledge of the domain and the ontology to be used [20]. The size of the ontology and the description document (in terms of the number of operations supported and their inputs and outputs) pose challenges when annotating Web services [6]. The chances of forming an invalid or incorrectly annotated WSDL/WADL document due to missing imports, tags or misspellings and incorrect references further deter the process. Currently, annotation tool support is lacking, thus compelling the user to go through the ontology and the WSDL/WADL document and find the appropriate annotations for each element that the user wishes to annotate. Then the user has to add the annotations to the WSDL/WADL document by manually editing the document. Thus,
a tool with some degree of automation that helps the user in annotating the Web services can greatly reduce the effort required.
CHAPTER 3
TECHNIQUES FOR ANNOTATING WEB SERVICES

Annotating the Web services description documents (WSDL/WADL) with concepts from the ontology requires adding model references and specifying the lifting and lowering schema mappings as described before. This can be done in two ways: either by manually editing the document or using a tool that helps the user in the annotation process. Often there are cases where users might not find the appropriate concepts they is looking to annotate the document with in the ontology they selected. In those cases the users have to model the concepts they thinks are required to be added to the ontology to facilitate annotation. Then finally add them to the ontology after careful examination.

3.1. Editing WSDL/WADL Documents

The manual editing process requires first finding the appropriate ontology that could be used for annotation. Secondly, the user needs to browse through the selected ontology and look for relevant concepts to annotate with. This might require loading the ontology in an editor/visualizer like Protégé [21]. The user then needs to obtain the URI for the concept and edit the WSDL/WADL document to add it as a model reference extension attribute. Manually editing the description documents poses a risk of creating invalid SAWSDL documents due to spelling errors, missing tags or missing imports. In addition to this, a large number of available Web services render this time-consuming manual process impractical.
3.2. Using GUI Based Annotation Tool

Since the manual approach of editing the WSDL/WADL documents cannot scale and is error prone, there is a need for an easy to use GUI based tool that can help the user in the annotation process. The Radiant tool developed by Gomadam, Verma, Brewer, Sheth and Miller [22, 23] was created as a plug-in to Eclipse [24] for annotating WSDL documents. Using this tool instead of manually editing the documents speeds up the annotation process. Still, the user has to go through the entire ontology to get the appropriate annotations for different elements of the description document. As real-world ontologies are large, it is almost impossible for the user to go through each and every concept in the ontology, in order to find the best match.

3.3. Automation – Providing Recommended Annotations

In order to solve the above problem, a tool that offers some degree of automation in annotating the WSDL/WADL documents is highly desired. Providing suggestions to the user with the concepts from the ontology that could possibly be used for annotating each element can greatly reduce the effort. This would require matching the concepts from the ontology to the elements in WSDL/WADL documents and providing the user with the best matches to choose from.

3.4. Recommendation Mechanisms

As mentioned before, an annotation tool with some degree of automation can greatly assist the user in the process of annotation. RadiantWeb employs a recommendation mechanism that suggests possible concepts from the ontology that can be used for annotation of different aspects of a description document. In semantic annotation of Web services, the main problems
encountered are finding the appropriate ontology to annotate the description document with and then finding the relevant concepts from the selected ontology. As discussed earlier, both of them are time consuming and providing recommendations in both the cases can considerably speed up the annotation process.

RadiantWeb solves the first problem by recommending possible ontologies to annotate a given WSDL/WADL document. The system makes use of the National Center for Biomedical Ontology (NCBO) ontology recommender [25] for achieving this. The recommender takes in text as input and returns a ranked list of ontologies that can be used for annotation. It ranks the ontologies using a semantic annotation based approach described in [26]. The NCBO recommender suggests the ontologies that are registered with the NCBO BioPortal [27] and Unified Medical Language System (UMLS) [28].

In order to handle the next problem, the system provides the user with best possible ontological concepts for annotation of each relevant element of the description document. The system calculates the match scores depending upon how well the ontological concept describes an element in a WSDL/WADL document. The system can recommend concepts in two different ways: (i) providing the best match for every element in the description document, and (ii) providing top-$k$ recommendations for a single requested element in the description document.

3.4.1. Problem Definition

The main problem in the process of recommending concepts for the elements in a description document is matching the ontological concepts with the elements from the description document. The concepts in the ontology are at semantic level described in terms of definitions, restrictions and relations between different concepts. However, the elements and
their documentation in the description document are at an XML level. The way we handle this problem is explained in the next section.

3.4.2. Matching WSDL/WADL elements to Ontological Concepts

Ontologies provide a richer way to model concepts by defining them in terms of restrictions, properties and constraints. Matching ontological concepts to the WSDL/WADL elements would require performing a string similarity between them. When performing a match, a common approach is to calculate the similarity between the names of the elements and labels of the concepts. Lack of any standard naming convention for naming the elements in the description document, can negatively hamper the results. For example, element "ktup" which is one of the inputs for ClustalW [29] stands for fast pairwise alignment word size and just using the element name in such cases is very unlikely to find the appropriate concept in the ontology. In order to tackle this issue, we make use of any available documentation for the element in WSDL/WADL document.

Using the element name & documentation and ontological concept label & definition, a Similarity score ($S$) is calculated, which is a match score between 0 and 1, where 1 signifies the perfect similarity between the ontological concept and the element in the description document. The similarity score is the weighted sum of definition similarity ($S_{definition}$), label similarity ($S_{label}$) and cross comparison between label and definition ($S_{definitionlabel}$).

$$
S = w_1 * S_{definition} + w_2 * S_{label} + w_3 * S_{definitionlabel}
$$

(formula 1)

where $w_1$, $w_2$ and $w_3$ are the normalized weights assigned for definition similarity, label similarity and cross comparison between label and definition, respectively. As the documentation of the element provides more information when compared to labels, we weigh the definition
similarity more than the other two. The weighing conditions for the different components are given in Table 1.

<table>
<thead>
<tr>
<th>Condition</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $S_{\text{definition}} = 0$</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>If $S_{\text{definition}} &gt; 0.4$</td>
<td>0.8</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>If $S_{\text{definition}} \leq 0.4$</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

1. **Label Similarity ($S_{\text{label}}$):** The label similarity component compares the label of an ontological concept to the name of a description document element. If the label is not present, the concept name is used instead. It is often possible that the element name and the ontological concept being compared might not have similar names, but they might mean the same. For example, the input parameter "ktup" in ClustalW means "alignment word size" in the ontology and the comparison between them gives a label similarity score of 0.052 & definition similarity score of 0.68. In order to handle such situations this component of the Similarity score ($S$) is assigned a lower weight. The Levenshtein distance [30] string metric is used to calculate the similarity between the names and this gives the score of similarity between 0 and 1.
2. **Definition Similarity** \( (S_{\text{definition}})\): Definition similarity contributes a significant portion towards the total Similarity score \( (S) \). Definition similarity calculates the similarity score between the documentation of the element and definition of the ontological concept. If one of them is not available a score of 0 is assigned and in such cases, only the other two components constitutes to the total similarity score. The Dice similarity [31] measure is used for this purpose. The Dice similarity is modified to fit our needs. The system preprocesses the definitions by removing all the stop-words and then uses the Porter stemmer algorithm [32] to trim the plural, -ing and -ed forms of the words found in the definition and then applies the dice similarity algorithm.

3. **Cross comparison between definition and label** \( (S_{\text{definitionlabel}})\): The cross comparison between the names of elements in the description document to the definition of concepts in the ontology and label in the ontology to the documentation in the WSDL/WADL element can also provide useful insights. As this has chances of yielding false positives, we assign lower weight for this component in the Similarity score calculation. Here first the stop-words from the ontological definition and the documentation of an element in WSDL/WADL are removed. Then Levenstein distance between the different words in the definition with the label is calculated and the maximum score is returned as the cross comparison score. \( S_{\text{definitionlabel}} \) score is the average of cross comparison score between definition of ontological concept to the name of an element in a WSDL/WADL document and the label of an ontological concept to a WSDL/WADL element documentation.

\[
S_{\text{definitionlabel}} = \frac{(s_1 + s_2)}{2} \quad \text{(formula 2)}
\]
where $s_1$ is the similarity score between definition of ontological concept to name of the description document element and $s_2$ is the similarity score between the label of an ontological concept to the documentation of a description document element.

3.5. RadiantWeb Annotation Tool

RadiantWeb is a Web application, which facilitates semantic annotation of Web service description documents (supporting both WSDL 1.1 and WADL) with concepts from an ontology with minimal effort by employing the recommendation mechanisms discussed before. The tool does not allow annotations from different ontologies at the same time however it supports annotations from multiple ontologies by loading them one after the other. The tool does not check for possible inconsistencies when annotating with different ontologies. The tool allows adding references to ontological concepts or to transformation documents that specify the mapping between parts of an XML schema and parts of an ontology schema. Schema mappings can play an important role in data mediation during Web service composition.
The interface to the RadiantWeb Annotation Tool is made up of two main panels, one on the left to load and display a Web service description document (WSDL/WADL) and one on the right for loading and browsing/searching the ontology. The tool provides support for OWL ontologies and uses the OWL API [33] for loading ontologies. The tool supports the annotation of porttypes, operations, inputs and outputs. The current version of the tool allows adding annotations to the WADL method and param elements. The tool supports bottom level annotation for complex types of the inputs and outputs.
The tool has undergone several modifications and improvements in an iterative process and finally resulted in a version that is easy to use. The initial version of the tool was not user friendly as it loads the document as XML and also does not provide any recommendations. As all the elements of the WSDL/WADL document need not be annotated, displaying the entire document in the form of XML makes it clumsy for users to annotate the document. To overcome this problem, the current version of the tool trims the complex structure of WSDL/WADL and presents the important information in a more readable way to the user. The current version provides recommendations for annotations and also aids the user to verify the recommendations by presenting the documentation of a selected description document element through a mouse-over and the definition of the selected ontological concept with the help of a definition panel.
CHAPTER 4
SYSTEM ARCHITECTURE

The architecture of the system is shown in the Figure 3. The system follows a slightly modified version of the Model View Controller (MVC) software architecture. The client-based controller handles the requests from the users and sends them to the view, which resides on the server and the view communicates with the model for obtaining the data and sends the formatted data to the controller. The controller will present the formatted data to the user using JavaScript.

Figure 3 shows the overall data-flow for the system. The initial data-flow starts by the user inputting the location of the description document to be annotated. This location is given either as path in the local file system or a URL in the Web. An Asynchronous JavaScript and XML (AJAX) call is made by the controller to the view telling it to store the description document in the server file system. Once the document is saved, it is loaded by sending a request to the model, which sends back the data formatted by the view and sent back to the controller. The controller then presents the description document in one of two ways: (i) a tree structure listing the service's operations as well as their inputs and outputs, or (ii) an XML view of the Web service description document.
4.1. Components

4.1.1. Controller

Controller is the component of the system that is responsible for handling all the interactions between the user and the view. The server loads the Web application and the
controller (coded in JavaScript) resides on the client Web page. The application always stays on this page throughout its lifecycle. The user requests are carried out through AJAX calls on the fly and the page is modified without having to actually reload the page or move to another page. As the application stays on one page, minor changes that do not require interaction from the server can be achieved easily using JavaScript.

This component handles different requests from the user like loading the WSDL/WADL document and ontology, recommending concepts to the description document elements, recommending ontologies for a particular WSDL/WADL document and exporting annotations. The interface to the page loaded is designed using jQuery [34] and jQueryUI [35] for the look and feel of the tool. Different interactive features of the tool include drag and drop, treeView, etc. The drag and drop is provided by the tool to facilitate manual annotation by dragging a concept from the ontology on to the WSDL/WADL part and the tree view is used for loading the description document and the ontology into a tree structure. The tool is tested on the following browsers: Mozilla Firefox, Google Chrome, Apple Safari and MS Internet Explorer.

4.1.2. View

The view component of the system is responsible for formatting the data that it gets from the model and sending it back to the controller. It gets the data in the form of a data structure or object and then the data structure or object is traversed accordingly to get the data and format it for display.

4.1.3. Model

In the MVC architecture, model is the component responsible for the implementation of the business logic. The model performs the actions specified by the view and returns the data. The
different subcomponents of the model are the WSDL/WADL manager, ontology manager, recommendation engine and ontology recommender.

Figure 4 illustrates how the model processes use cases in which the user wishes to annotate an operation, input or output. A request is sent from the client to the view, which in turn requests the model to recommend the candidate annotations. The candidate annotations are found by performing text matching between the description document (WSDL/WADL) and the concepts (OWL classes) in the selected ontology.

Figure 4: Model Component of the System
• WSDL/WADL Manager

The WSDL/WADL manager is responsible for loading the description document either from a local file system or a document that is hosted online. The WSDL/WADL manager has the following sub-components: an XML schema parser based on DOM, WSDL/SAWSDL parser based on Web Service Description Language for Java (WSDL4J) [36] and Java-based Document Object Model (JDOM) [37] in case of WSDL. As the only parser we found for WADL predated the latest revision of the standard, we developed our own parser for WADL using JDOM. WSDL4J parses the WSDL document and retrieves all the elements, except the XML schema elements. Therefore, to retrieve the XML schema elements referenced in the WSDL document, the XML schema parser is used.

The code snippet shown in Figure 5 explains how to retrieve the input type of given a Web service's operation name (e.g., the *run* operation from the *WUBlast* Web service). In general, the type of input (assuming it is non-recursive) could be expressed as a Directed Acyclic Graph (DAG) or as a Tree. Currently, in this work we are only performing bottom-up annotation. Hence, we only annotate the leaves of the DAG/Tree. In order to retrieve all of the leaf level elements, we navigate from the operation to its input's message. We then obtain the leaf level elements for each message part. This involves navigating through all the existing complex types that are part of the input.
List<Element> elements = new List<>();
Operation op = getOperation("run");
Input in = op.getInput();
Message msg = in.getMessage();
for (p: msg.getParts())
    elements.add = getLeafNodes(p)

Figure 5: Code snippet to get the leaf nodes of a given operation

• Ontology Manager

The ontology manager is responsible for loading the ontology and can load ontologies from a local file system or an ontology that is hosted on the Web. The ontology manager has different sub-components, such as the OWL handler and ontology search. The handler is used for loading the ontology as a tree structure showing the class hierarchy and is implemented using the OWL API [33]. There are two main APIs available for creating, manipulating and serializing ontologies: Jena API and OWL API. Jena is mostly flexible for dealing with data graphs and supports SPARQL queries on Resource Description Framework (RDF). OWL API on the other hand does not support the querying of RDF data and is mostly support OWL constructs like support for OWL 2. As this project is more focused in using the ontology schema for annotation rather than the data, OWL API is used.

The ontology manager uses a structural reasoner available in OWL API to infer the class hierarchy while loading the ontology. This is because in some cases the class hierarchy in ontologies is not asserted but inferred. The structural reasoner employed here performs limited checking and therefore runs relatively quickly compared to other reasoners. The search sub-component helps the user in finding the concepts that a user is
looking for without having to browse through the entire ontology. The search uses the Levenshtein distance metric to calculate the similarity between the user query and the labels of ontological concepts and presents the user with the top-$k$ results along with their definitions.

- Recommendation engine

The recommendation engine is responsible for suggesting most probable concepts for annotations by performing the matching of the ontological concepts to the elements of WSDL/WADL documents. Traditionally, the entire ontology should be searched in order to find the relevant concepts, but this greatly increases the time for recommendation, as the size of the ontologies may be huge. In order to speed up the recommendation process, we search only the subparts of the ontology where the operations and data could reasonably be found. When loading a new, so far unused ontology, the user must provide key super-classes for the operation, inputs and outputs (often the inputs and outputs may use the same super-class). For example in Figure 6, the code snippet is used to rank high scoring sub-concepts of the "data item" concept. These are concepts that match one of the leaf elements, $e_i$, of the input to the run operation of the WUBlast Web service. Given the returned ranked list, the user may select one or more of them as the annotations for $e_i$. 


Element e_i = elements.get(i)
for (cls: OntologyHandler.getSubClasses("data item")){
    String defn = pre(OntologyHandler.getDefinition(cls))
    String label = OntologyHandler.getLabel(cls)
    s_defn = compareDice(defn, e_i.getDocumentation())
    s_label = compareLevenshtein(label, e_i.getName())
    s_cross1 = compareCross(defn, e_i.getName())
    s_cross2 = compareCross(e_i.getDocumentation(), label)
    score = computeScore(s_defn, s_label, s_cross1, s_cross2)
}
return high-ranking matches as an ordered list

In order to perform the matching, the recommendation engine requires the name and documentation of the WSDL element to be annotated (e.g., input parameter "sequence" from WU-Blast's run operation), as well as, the definitions and labels of concepts from the ontology. The recommendation engine communicates with the ontology manager and obtains the key sub-classes of the ontology for the operation, inputs and outputs. It also communicates with the WSDL/WADL manager to get all the elements and operations of the description document. The string metrics sub-component calculates the Similarity scores ($S$) for the retrieved ontological concepts and ranks them according to their scores. The ranked list is then returned to the view, which sends it to the controller for presentation to the user. Then the user needs to pick the appropriate concept(s) with which to annotate the WSDL element. The ontology should provide proper definitions and the WSDL document should be well documented. If the user does not find an appropriate concept in the recommended list, they may (i) use the ontology search feature to find the better concept, (ii) navigate the ontology, or (iii) use the special concept called un-modeled term.
• Ontology Recommender

The ontology recommender is used to help the user in choosing an appropriate ontology for annotation of a particular description document. The ontology recommender uses the NCBO recommender service that searches for a relevant ontology taking text as input. The ontology recommender contacts the WSDL/WADL manager to retrieve the documentation from the WSDL/WADL document and passes it to the NCBO recommender as input. The NCBO ontology recommender gives a list of ontology ids and their names, as output. In order to get the location of the ontology, another REST Web service [38] from NCBO is invoked, which takes the ontology id as input and gives the details of the ontology. The system filters out the ontologies that are not in OWL format and returns the top-$k$ results to the controller, which presents it to the user. The user can choose one of the ontologies and it will be loaded for annotation of the WSDL/WADL document.

• Repository

After the user is satisfied with the annotations added so far, they may save the annotated document in the repository. The repository, as a simple replacement for UDDI\(^1\), can be used for retrieving annotated document at some later time by the user as well as used for discovery of Web services. The system uses an XML database called eXist [39] to implement the repository.

4.2. Usage

\(^1\) The first version of Radiant used jUDDI to store Web service metadata.
There are six approaches for using RadiantWeb for annotating documents. Using the first approach, the user requests for a recommendation for each element in the description document and the system gives the best-matched concept for each element in the WSDL/WADL document. Now the user has to approve the terms for annotation. If the user is not satisfied with the recommended concept for a specific element, then the next approach would be to request the recommended concepts for that particular element and the system returns the top-\(k\) recommended concepts. The user then has to pick a concept that he thinks is appropriate for annotation. If the user is still not able to find the appropriate concept for annotation, then he has to move to the next approach of manual annotation. The manual annotation is performed by searching for the appropriate concept and annotating the WSDL/WADL element with the help of a drag and drop facility provided by the system. For searching the ontology, the search mechanism provided by the ontology manager can be used. Now the difficulty would be in finding what to key in for the search. In one approach the user can use the important words from the definition of the element for searching and in the other approach the user can enter keywords based on the user's knowledge about the element. The user will be presented with the top-\(k\) concepts as the result from the search and the user can check for the concepts that are appropriate for annotation. If none of the above approaches suffices, then the only approach left to follow is to browse through the ontology manually by opening the class hierarchy in the ontology viewer and then find the appropriate concept, which can be very challenging. Even though the tool greatly aids the user in the annotation process, it does not remove the need for domain knowledge by the user, as the user has to approve the terms for annotations.
CHAPTER 5
RELATED WORK

Our work focuses on tool support for annotation of Web services with a certain level of automation, which substantially reduces the effort required to annotate Web services. Even though there has been much research on annotating Web documents, there are only a few research efforts aimed at semantic annotation of Web services in a semi-automatic way. We discuss three distinct research efforts that aim at annotating Web services.

One of the efforts is the work in [40] that provides a tool called ASSAM for semi-automatic annotation of Web services. The tool utilizes machine-learning techniques to obtain annotation recommendations for un-annotated services. It uses existing annotated documents, assuming a sufficient number of such documents that are already annotated, as a training-set for the algorithms. The authors have designed a classification algorithm, Iterative Relational Classification Algorithm, which use the intrinsic and extrinsic features of a Web service for classification of the inputs, outputs and operations. The annotations here are exported as OWL-S groundings.

As part of the METEOR-S project, a paper by Patil et al. [6] studied semi-automatic annotation of Web services. This work addresses the structural matching issues involved in comparing ontology concepts to WSDL/XSD elements. It converts both the WSDL and ontology into common structures called SchemaGraphs based on pre-defined rules and applies similarity algorithms on these graphs. The work uses different string similarity metrics, but just on the names of the WSDL elements and assumes that good naming practices are utilized. In some
cases, ontologies use ids, such as OBIws_0000088, for the concept name. Therefore, RadiantWeb makes use of the label of the ontological concept and documentation from the description document, so it works even if the names do not match. The tool exports the annotations in a SAWSDL document. The work by Patil et al. focuses only on the annotation of inputs and outputs, whereas RadiantWeb supports annotation of inputs, outputs, operations and porttypes.

The work in [41] presents a methodology for annotating a large number of Web services and does not provide tool support. The methodology involves an iterative process and extends an ontology as and when needed, so it includes an incremental ontology building process. Therefore, the target audience should be highly skilled annotators with knowledge of ontology building rather than just an ordinary user. However, our work focuses on providing an easy to use tool, so that an ordinary user can make use of the tool with minimal effort. Their work makes passes over sets of WSDL files annotating the most frequent elements and creates heuristics. A heuristic rule for the concept C is of the form $C \leftarrow \{e_1, e_2, ..., e_n\}$ where $e_i$ is a regular expression used to match against WSDL element names (e.g., $\text{Student\_ID} \leftarrow \{\text{SID, Stud\_ID, student*identifier}\}$); these rules are used to map the elements to available concepts. Clearly, such rules are inherently domain specific. Many of the steps in this work are manual even though the authors call it semi-automated.
CHAPTER 6
EVALUATION

For the purpose of evaluation of the tool, we have chosen to annotate 11 Web Services from the European Bioinformatics Institute (EBI) [42] in the bio-informatics domain with the concepts from an enriched version of the Ontology for Biomedical Investigations (OBI) [43]. The Web services used for the evaluation are NCBIBlast, WUBlast, PSIBlast, PSISearch, FASTA, FASTM, ClustalW, Muscle, Kalign, Mafft and DBClustal. The Web services perform different bio-informatics tasks. Each of these services has 6 different operations performing different tasks based on the service functionality. In total there are 252 input elements, 176 output elements and 66 operations for all these Web services. The OBI ontology describes biological and clinical data and has been enriched in order to fulfill the needs of annotating Web services in the biomedical domain.

The accuracies of the top-1, top-3, top-5, and top-7 recommendations provided by the recommendation engine are calculated for two cases: (i) when both definition and label are considered and (ii) when only labels are considered. The accuracy of the best (top-1) recommendations for case (i) is 66.51%. The accuracy increases to 70.19% for the top-3, 82.75% for the top-5 and finally reaches the maximum of 84.23% for top-7 recommendations. The accuracy is also calculated for case (ii), which shows significantly lower accuracy when compared to case (i). The accuracy for the best (top-1) recommendations is 25.86%, top-3 is 42.11 %, top-5 is 44.58% and top-7 is 45.07%. The graph in Figure 7 shows the above statistics.
for both the cases. From the graph it is evident that considering the documentation from the WSDL and definitions from the ontology has improved the suggestions greatly.

Figure 7: Comparing the accuracy of recommended concepts in both cases

Figure 8: Graph showing the accuracy of recommendations for each service in both cases
The accuracy is calculated for each and every Web service for both cases and the graph indicating the same is shown in Figure 8. The accuracy for inputs, outputs and operations has been calculated individually and is summarized in Table 2 and represented as a graph in Figure 9.

Table 2: Accuracy (%) of inputs, outputs and operations for both the cases

<table>
<thead>
<tr>
<th></th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
<th>Top-7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Label &amp; definition</td>
<td>Label &amp; definition</td>
<td>Label &amp; definition</td>
<td>Label &amp; definition</td>
</tr>
<tr>
<td>Inputs</td>
<td>62.26</td>
<td>15.47</td>
<td>73.8</td>
<td>32.93</td>
</tr>
<tr>
<td></td>
<td>80.95</td>
<td>34.52</td>
<td>83.33</td>
<td>35.31</td>
</tr>
<tr>
<td>Outputs</td>
<td>75</td>
<td>25</td>
<td>75</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>87.5</td>
<td>50</td>
<td>87.5</td>
<td>50</td>
</tr>
<tr>
<td>Operations</td>
<td>50</td>
<td>66.66</td>
<td>50</td>
<td>66.66</td>
</tr>
<tr>
<td></td>
<td>75.75</td>
<td>75.75</td>
<td>83.33</td>
<td>75.75</td>
</tr>
</tbody>
</table>

Figure 9: Graph showing accuracy for inputs, outputs and operations in both cases
CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This thesis discusses the need for semantic annotations for Web services and the problems involved in achieving the same. It presents various methods for annotating a description document and the problems in each method and how they are solved by other methods. We have developed a tool to facilitate semantic annotation of Web services with support for both SOAP (WSDL) and REST (WADL) services that can operate at different levels of automation. We discuss the problem of matching ontological concepts to description document elements and present our methodology to achieve this. The tool employs different syntactic matching mechanisms and utilizes available documentation in the WSDL/WADL and definitions in the Ontology for matching the ontology concepts to the WSDL/WADL elements. The tool also supports adding schema mapping annotations. The matching techniques employed have been evaluated for accuracy of the recommendations provided by the system. The evaluation shows that the use of documentation of the description document for calculating the similarity scores has considerably improved the accuracy of the recommendations.

There are many ways in which the RadiantWeb annotation tool can be improved that are left for future work. The tool current version of the tool does not provide support for WSDL 2.0, which can be supported in future. The validation of the Web services can be provided by invocation of Web services. Currently, when calculating the similarity score, the system does not take into account if the two words or synonyms or not which can be facilitated in the next version with the help of Wordnet [44]. The system annotates only the leaf nodes of inputs and
outputs leaving the intermediate nodes so, annotating the whole structure of inputs or outputs should be implemented. Setting the weights for the similarity score using machine-learning algorithms. The matching speed can be improved by using indexing techniques like Lucene. Schema graphs proposed by [6] can be used to improve the accuracy. During the calculation of similarity score even though the labels and definition are considered, in some cases the concept class identifier can also be meaningful and so a component based on this information can also be added.
REFERENCES


[8] "Web Service Description Language (WSDL)," http://www.w3.org/TR/wsdl20/,
[9] "Web Application Description Language (WADL)," [http://www.w3.org/Submission/wadl/].


[42] "European Bioinformatics Institute (EBI)," http://www.ebi.ac.uk/.


APPENDIX A

USER GUIDE

The user guide gives the details of how to use the RadiantWeb tool for annotation. The tool has two panels the WSDL viewer and the ontology viewer. The WSDL/WADL viewer loads the description document in a tree structure trimming all the complex structure of the WSDL/WADL where as the ontology Viewer loads the ontology as tree structure of class hierarchy. The ontology viewer loads the ontology as a tree structure of its class hierarchy. The following step will explain how to use the tool.

Step 1: Load the WSDL/WADL document: The WSDL/WADL viewer is used to load the description document and the input can be given as a file or URL of the document that is hosted online. Input the file through the browse button or URL of the document in the text box and then click on load WSDL/WADL button to load the document.

![WSDL/WADL Viewer](image)

Figure 10: Description document loader

Step 2: Load Ontology: The ontology viewer is used to load the OWL ontologies from local system or from the Web. Apart from this, the user can also request the ontology recommender to provide the suggested ontologies to annotate a particular WSDL/WADL is loaded. Once the user selects the ontology it will be loaded and can be used for annotation. The
system supports annotation with one ontology at a time. If the user wants to annotate with a different ontology, then he needs to load it.

![Ontology Viewer](image)

Figure 11: Ontology loader

**Step 3: Annotate the Web Service.** There are five approaches to achieve this.

1. The first and the easiest approach would be to use the recommend terms button at the top of the loaded document. The best recommendation is provided for each part of the Web service below it in red color indicating that the annotation should be approved. If the user is satisfied with recommendation the user can approve the recommendation by clicking the tick icon at the front of the annotation. Otherwise the user could reject the recommendation by clicking on the cross icon next to the tick icon.
2. If the user is not satisfied with the recommendation for a particular element in the document, then the user can use the next approach of recommending the top-$k$ ontological concepts for annotation of a particular element. This is done using the "Recommend Terms" button on that particular element. The results are displayed in a dialog box and then the user can choose the appropriate terms to be annotated from the dialog box.
3. If the user is still not able to find appropriate concepts from the top-$k$ recommended concepts then the user can switch to the approach of using manual annotation using the ontology search. Now the user searches for an appropriate concept from the loaded ontology and this is done with the search mechanism provided by the ontology viewer and the user can key in the important words from the definition.

The results are presented in a dialogue box along with its definition and when the user finds an appropriate concept for annotation, the user can simply double click
on that concept and the concept will be located in the ontology class hierarchy available in the ontology viewer.

![Figure 15: Ontology search results](image)

Then the user has to drag and drop the selected ontological concept on the description document element the user want to annotate. If the user cannot find the appropriate concept even now next approach should be employed.

4. This approach is basically modifying the search employed in the approach 3. Here the user keys in the terms that he thinks best describe the element into the ontology search field. Once the results are returned he can choose a concept from the ontology and then annotate it to the WSDL/WADL element with the drag and drop facility provided by the tool for manual annotation.

5. Finally if the approach 4 also fails the only one possible way to achieve that is by browsing thorough the ontology, checking the definition of the concepts and finding the appropriate concepts for annotation. If the user thinks that a particular concept is appropriate for annotation of the element the user is looking for, then he can do so by using the drag and drop feature. For this approach the user is required to have a comprehensive knowledge about the ontology.
Step 4: After the process of adding annotations is completed the annotations are saved into a file and the file is saved either to a repository that is an XML database, exist or give the user the ability to download the file.

DEPLOYMENT GUIDE

The guide gives the instructions to the users how to create the war file from the project and how to deploy the war file on a server. It explains the deployment of the application in Tomcat server.

Step 1: Build the war file: The war file can be built using the ant build script available in the project. Open a command prompt or terminal and navigate to the project home folder and build the war file using the following command:

$ ant war

This above command cleans the old files, compile the code and place them in the WEB-INF/classes folder and package everything in to a war file and place it in dist folder in the project folder.

Step 2: Deploy the war file: Then the user has to deploy the war file. In Tomcat this can be achieved in two ways:

1. Copy the war file to the webapps folder of the tomcat installation directory and it will be auto deployed.

2. Use the tomcat manager to deploy the war file through Web browser. Open Tomcat in the browser and then open the tomcat manager in it. The Tomcat Web Application manager can deploy a Web application by providing it with the war file.
APPENDIX B

SAMPLE ANNOTATED DOCUMENT

<wsdl:types>
<xsd:schema xmlns="http://soap.jdispatcher.ebi.ac.uk" attributeFormDefault="unqualified" elementFormDefault="unqualified" targetNamespace="http://soap.jdispatcher.ebi.ac.uk">
<xsd:complexType name="InputParameters">
<xsd:annotation>
<xsd:documentation xml:lang="en">Input parameters for the tool</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element minOccurs="1" maxOccurs="1" name="program" nillable="false" type="xsd:string" sawsdl:modelReference="http://purl.obolibrary.org/obo/OBIws_0000088">
<xsd:annotation>
<xsd:documentation xml:lang="en">Program [The BLAST program to be used for the Sequence Similarity Search.]</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element minOccurs="0" maxOccurs="1" name="exp" nillable="true" type="xsd:string" sawsdl:modelReference="http://purl.obolibrary.org/obo/OBIws_0000082">
<xsd:annotation>
<xsd:documentation xml:lang="en">Expectation value threshold [Limits the number of scores and alignments reported based on the expectation value. This is the maximum number of times the match is expected to occur by chance.]
</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element minOccurs="0" maxOccurs="1" name="alignments" nillable="true" type="xsd:int" sawsdl:modelReference="http://purl.obolibrary.org/obo/OBIws_0000075">
<xsd:annotation>
<xsd:documentation xml:lang="en">Alignments [Maximum number of match alignments reported in the result output.]</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element minOccurs="0" maxOccurs="1" name="scores" nillable="true" type="xsd:int" sawsdl:modelReference="http://purl.obolibrary.org/obo/OBIws_0000073">
<xsd:annotation>
<xsd:documentation xml:lang="en">Scores [Maximum number of match score summaries reported in the result output.]</xsd:documentation>
</xsd:annotation>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
</wsdl:types>
<wsdl:portType name="JDispatcherService">
<wsdl:operation name="run">
<wsdl:documentation>Submit an analysis job</xsd:documentation>
<wsdl:input name="runRequest" message="tns:runRequest"/>
<wsdl:output name="runResponse" message="tns:runResponse"/>
</wsdl:operation>
</wsdl:portType>
</wsdl:definitions>

Figure 16: Fragment of WUBLAST.sawSDL
## APPENDIX C

### JAVA DOC OF THE PACKAGES

<table>
<thead>
<tr>
<th>Packages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>edu.uga.radiant.annotation</code></td>
<td>Provides classes for loading the WSDL document</td>
</tr>
<tr>
<td><code>edu.uga.radiant.loadWSDL</code></td>
<td>Provides classes necessary for loading the WSDL document.</td>
</tr>
<tr>
<td><code>edu.uga.radiant.ontology</code></td>
<td>Provides classes for parser based on OWL API for parsing the ontology, ontology search and ontology loader.</td>
</tr>
<tr>
<td><code>edu.uga.radiant.stringmetrics</code></td>
<td>Provides classes for calculating different string metrics employed in the similarity score.</td>
</tr>
<tr>
<td><code>edu.uga.radiant.suggestion</code></td>
<td>Provides classes for calculating the similarity score, ontological terms recommender, ontology recommender.</td>
</tr>
<tr>
<td><code>edu.uga.radiant.util</code></td>
<td>Provides different utility classes utilized by the system.</td>
</tr>
<tr>
<td><code>edu.uga.radiant.wadlparser</code></td>
<td></td>
</tr>
<tr>
<td><code>edu.uga.radiant.wsdlparser</code></td>
<td>Provides classes for parsing the WSDL documents.</td>
</tr>
</tbody>
</table>