METEOR-S Web Service Annotation Framework

Abhijit A. Patil
Swapna A. Oundhakar
Amit P. Sheth
Wright State University - Main Campus, amit@sc.edu
Kunal Verma
Wright State University - Main Campus

Follow this and additional works at: https://corescholar.libraries.wright.edu/knoesis
Part of the Bioinformatics Commons, Communication Technology and New Media Commons, Databases and Information Systems Commons, OS and Networks Commons, and the Science and Technology Studies Commons

Repository Citation

This Conference Proceeding is brought to you for free and open access by the The Ohio Center of Excellence in Knowledge-Enabled Computing (Kno.e.sis) at CORE Scholar. It has been accepted for inclusion in Kno.e.sis Publications by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.
ABSTRACT
The World Wide Web is emerging not only as an infrastructure for data, but also for a broader variety of resources that are increasingly being made available as Web services. Relevant current standards like UDDI, WSDL, and SOAP are in their fledgling years and form the basis of making Web services a workable and broadly adopted technology. However, realizing the fuller scope of the promise of Web services and associated service-oriented architecture will require further technological advances in the areas of service interoperation, service discovery, service composition, and process orchestration. Semantics, especially as supported by the use of ontologies, and related Semantic Web technologies, are likely to provide better qualitative and scalable solutions to these requirements. Just as semantic annotation of data in the Semantic Web is the first critical step to better search, integration and analytics over heterogeneous data, semantic annotation of Web services is an equally critical first step to achieving the above promise. Our approach is to work with existing Web services technologies and combine them with ideas from the Semantic Web to create a better framework for Web service discovery and composition. In this paper we present MWSAF (METEOR-S Web Service Annotation Framework), a framework for semi-automatically marking up Web service descriptions with ontologies. We have developed algorithms to match and annotate WSDL files with relevant ontologies. We use domain ontologies to categorize Web services into domains. An empirical study of our approach is presented to help evaluate its performance.

Categories and Subject Descriptors
D.2.12 [Software Engineering]: Interoperability – data mapping, interface definition languages, D.3.1 [Programming Languages]: Formal definitions and Theory – semantics, H.3.5 [Information Storage and Retrieval]: Online Information Services – data sharing, web-based services

General Terms
Algorithms, Experimentation.

Keywords
Semantic Web services, WSDL, Ontology, semantic annotation of Web services, Web services discovery

1. INTRODUCTION
The Web services are the latest attempt to revolutionize large scale distributed computing. With XML based standards like UDDI, WSDL, and SOAP, they are touted as the tools for universal connectivity and interoperability of applications and services. With the growing popularity of Web services, there arise issues of finding relevant services, especially with the possibility of the existence of thousands of Web services. We envision Web services as being initially applied more to address B2B/EAI challenges, rather than B2C services. In this context, Web services will be used as part of larger Web processes that result from Web services composition. Current standards have focused on operational and syntactic details for implementation and execution of Web services. This limits the search mechanism for Web services to keyword-based searches. Consider a scenario, where a user may want a Web service that takes “weather station code” as input and gives “Atmospheric conditions” as output. The current search mechanism at a popular Web service repository like Salcentral.com allows only keyword searches. Searching for the keyword “weather” gives about 3% of the total Web services in that repository. It returns all Web services, which have weather, mentioned in their description. The user has to manually analyze WSDL files to find the appropriate service. Five years from now when we are expected to have thousands of services, current syntactic search along with manual intervention would be untenable. Research in the Semantic Web area has shown that annotation with metadata can help us solve the problem of inefficient keyword based searches in the current web (which is based on HTML, HTTP, and URI’s). This concept of annotation can be extended to Web services to envision Semantic Web services. Semantically described services will enable better service discovery, allow easier interoperation and composition of Web services.

Several approaches have already been suggested for adding semantics to Web services. Semantics can either be added to currently existing syntactic Web service standards like UDDI or WSDL [1] or services can be described using some ontology based description language like DAML-S [2]. The common factor in most of these approaches is relating concepts in Web services to domain specific ontologies. This relating and tagging of descriptions with concepts in ontologies is referred to as annotation. While significant research has been done on what to annotate, there has been little work on how to annotate. Current research of Web service annotation largely focuses on manual annotation [3] that poses several problems. The first problem is that of finding the relevant ontology or ontologies. In manual annotation, the burden of choosing the relevant ontology or ontologies lies with the user. This significantly increases the pre-match effort as the user has to browse through the available ontologies to find suitable domain ontology or ontologies (since a Web service may span more than one domain and may have to be mapped to a number of ontologies). The second problem arises because of the size of the Web service description and the size of the ontology or vocabulary. As Web service descriptions grow larger (e.g., even a modest Web service “GlobalWeather” by CapeScience has 53 different elements and 55 different values of the parameters adding up to 108 concepts), the potential number of concepts in the Web service increase manifold. Furthermore
the vocabularies, taxonomies, or ontologies used for annotation could also be very large with correspondingly large number of concepts (e.g. the world-fact-book ontology contains more than 1100 concepts). Notice that real average world ontologies have been reported to exceed over 1 million instances [4]. Hence finding the appropriate ontological concepts to match to WSDL concepts can be a very tedious task. As we progress towards a Web of services, the number of services is likely to be in thousands or even more [5]. Given these factors, it is necessary to have a scalable and semi-automated way of annotating Web services with real world ontologies.

A key enabling capability that can address the above scalability challenge is annotation with as much automation as possible without losing quality. We present a framework, METEOR-S Web service Annotation Framework (MWSAF), to semi-automatically annotate WSDL descriptions of the services with relevant ontologies. MWSAF is a part of an ongoing project, METEOR-S, an effort to create Semantic Web processes, at the LSDIS lab, University of Georgia. We have implemented a number of algorithms to match concepts in WSDL files to ontologies.

We describe the architecture, implementation, and working of the MWSAF in this paper. The main contributions of our work are

- Addressing the need for semantics in the Web services framework, and providing a detailed approach that identifies four types of semantics for describing Semantic Web services
- Identifying the technical challenges in (semantic) annotation of Web services.
- Implementing algorithms for Semantic Annotation and categorization of Web services
- Empirical testing of semantic annotation of Web services

The rest of the paper is organized as follows: Section 2 describes the four types of semantics involved in the Web services framework. Section 3 describes the implementation and matching issues. The architecture is discussed in Section 4, Section 5 discusses the empirical study. Section 6 lists the related works. We conclude in Section 7 and give an outline for future work.

2. METEOR-S - ADDING SEMANTICS TO WEB SERVICES INDUSTRIAL STANDARDS

There have been several attempts to add semantics to Web services [6][1]. The METEOR-S project at the LSDIS Lab, UGA attempts to add semantics to the complete Web process lifecycle by providing constructs for adding semantics to current industry standards. We believe, our approach is more pragmatic, than other top down approaches [2], which require developing new standards with no tangible benefits over our approach. We identify the four categories of semantics in the complete Web process lifecycle [7].

- Data Semantics (semantics of inputs / outputs of Web services)
- Functional Semantics (what does a service do)
- Execution Semantics (correctness and verification of execution )
- QoS Semantics (performance/cost parameters associated with service)

Covering the complete lifecycle of Web services involves adding the four categories of semantics to different layers of the Web service stack [8]. The service description layer of the stack provides the information necessary for invoking Web services. WSDL is the de facto standard for this layer. However, WSDL descriptions are syntactic and do not explicate the semantics of the service providers. METEOR-S provides a mechanism to add data, functional and QoS semantics to WSDL files. MWSDI provides an infrastructure to leverage data, functional and QoS semantics by enhancing UDDI [9]. The top layer i.e. the flow layer deals mainly with service composition. A comprehensive framework for Semantic Web service composition is provided in MWSCF [10]. This paper concentrates on a semi-automated approach for adding data semantics to WSDL files.

Figure 1 gives an overview of these four types of semantics and different stages of Web process lifecycle development.

![Figure 1. Four types of Semantics in Web services](image)

3. IMPLEMENTATION AND MATCHING ISSUES

Expressiveness of ontologies and the XML schema used by WSDL are significantly different because of the different reasons behind their development [11]. The XML schema is used in WSDL descriptions to provide a basic structure to the data exchanged by the Web service. It therefore provides a minimal containment relationship using the complexType, simpleType and element constructs. On the other hand ontologies are developed to capture real world knowledge and domain theory [12]. Therefore the languages used to describe the ontologies, model the real world entities as classes (concepts) and their properties. They also provide the named relationships between different concepts and properties, making it easier to model entities in the real world more expressively. For example, consider the real world phenomenon “Snowfall which is caused by extreme low temperatures”. An ontology can very well describe this phenomenon because it can have concepts “extreme low temperatures” and “Snowfall” and relate the two with the named relationship “causes”. A WSDL schema can have the elements “extreme low temperatures” and “Snowfall” but since there is no support for named relationships, it cannot represent this phenomenon.
3.1 SchemaGraphs

The difference in expressiveness of XML schema and ontology makes it very difficult to match these two models directly. A possible solution to this problem is to convert both the models to a common representation format to facilitate better matching. We have used this approach and devised a representation format called SchemaGraph. The advantage of using SchemaGraph is that ontology can be in any language like DAML, RDF-S, or OWL etc. A SchemaGraph is a set of nodes connected by edges. We use conversion functions to convert both XML schema and ontology to SchemaGraphs.

The conversion function used for converting XML schema to SchemaGraph is WSDL2Schema and uses the conversion rules specified in Table 1.

### Table 1. XML Schema to SchemaGraph conversion rules

<table>
<thead>
<tr>
<th>XML schema Construct</th>
<th>SchemaGraph representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplexType</td>
<td>Node</td>
</tr>
<tr>
<td>Elementary XML Data Type Element defined under complexType</td>
<td>Node and an Edge between complexType node and this node with name “hasElement”</td>
</tr>
<tr>
<td>ComplexType Data Type Element defined under complexType</td>
<td>Edge</td>
</tr>
<tr>
<td>SimpleType</td>
<td>Node</td>
</tr>
<tr>
<td>Values defined for simple types</td>
<td>Node and edge between simpleType and this node with name “hasValue”</td>
</tr>
<tr>
<td>Elements</td>
<td>Nodes</td>
</tr>
</tbody>
</table>

**Example**

```xml
<daml:Class rdf:ID="WindEvent">
  <rdfs:comment>Superclass for all events dealing with wind</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#WeatherEvent" />
</daml:Class>
<daml:Property rdf:ID="windDirection">
  <rdfs:label>Wind direction</rdfs:label>
  <rdfs:domain rdf:resource="#WindEvent" />
  <rdfs:range rdf:resource = "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:Property>
<daml:Property rdf:ID="windSpeed">
  <rdfs:label>Wind speed</rdfs:label>
  <rdfs:domain rdf:resource="#WindEvent" />
  <rdfs:range rdf:resource="#Speed" />
</daml:Property>
```

WSDL2Schema is the conversion function used for creating SchemaGraph representation of the ontology. The set of conversion rules used for this are listed in Table 2.

### Table 2. Ontology to SchemaGraph conversion rules

<table>
<thead>
<tr>
<th>Ontology representation</th>
<th>SchemaGraph representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Node</td>
</tr>
<tr>
<td>Property with basic datatypes as range (Attribute)</td>
<td>Node with edge joining it to the class with name “hasProperty”</td>
</tr>
<tr>
<td>Property with other class as range (Attribute)</td>
<td>Edge between the two class nodes</td>
</tr>
<tr>
<td>Instance</td>
<td>Node with edge joining it to the class with name “hasInstance”</td>
</tr>
<tr>
<td>Class – subclass relationship</td>
<td>Edge between class node to subclass node with name “hasSubClass”</td>
</tr>
</tbody>
</table>

**Example**

```
<xsd:complexType name="Direction">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" nillable="true" name="compass" type="xsd1:DirectionCompass" />
    <xsd:element maxOccurs="1" minOccurs="1" name="degrees" type="xsd:int" />
  </xsd:sequence>
</xsd:complexType>
```

Once both the ontology and the XML schema are represented in a common SchemaGraph representation, we apply our matching algorithm to find the mappings between them. Once a concept is matched against all the concepts in an ontology, the best mapping needs to be picked out for annotation. In the next few sections we present our algorithm to calculate the match between two SchemaGraphs.

### 3.2 Mapping two concepts

Every concept from the WSDL SchemaGraph is compared against concepts from the ontology SchemaGraph. The function findMapping listed in Table 3 returns the mapping between a WSDL and ontology concept pair which consists of wc, oc, (WSDL concept and ontology concept) and MS i.e. Match Score.

---

1 Currently we use ontologies represented using RDF-S and subset of DAML+OIL.
The MS is composed of two different measures Element Level Match (ElemMatch) and Schema level match (SchemaMatch). ElemMatch provides the linguistic similarity of two concepts whereas SchemaMatch takes care of structural similarity. The MS is calculated as the weighted average of ElemMatch and SchemaMatch as shown in Equation 1.

\[
MS = \frac{w_1 \times \text{ElemMatch} + w_2 \times \text{SchemaMatch}}{w_1 + w_2}
\]

where, \((0 \leq w_1 \leq 1)\) \((0 \leq w_2 \leq 1)\)

### Equation 1. Formula for Calculating Match Score (MS)

Weights \(w_1\) and \(w_2\) indicate the contribution of Element level match and Schema level match respectively in the total match score. If two concepts have a matching structure then more weightage should be given to the SchemaMatch. If a WSDL concept does not have any structure then the SchemaMatch should not be considered. Based on these conditions the values of \(w_1\) and \(w_2\) are changed as shown in Table 4.

### Table 4. Weight values for calculating MS

<table>
<thead>
<tr>
<th>Condition</th>
<th>(w_1)</th>
<th>(w_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>WSDL concept is leaf node</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SchemaMatch (&gt; 0.9), ElemMatch (&lt; 0.9)</td>
<td>0.1</td>
<td>0.9</td>
</tr>
<tr>
<td>SchemaMatch (&gt; 0.75), ElemMatch (&lt; 0.75)</td>
<td>0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>SchemaMatch (&gt; 0.65), ElemMatch (&lt; 0.65)</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>SchemaMatch (&lt; 0.5), ElemMatch (&gt; 0.5) and WSDL Concept is of SimpleType</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SchemaMatch (&lt; 0.5), ElemMatch (&lt; 0.5) and WSDL Concept is of SimpleType</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

#### 3.2.2. Schema level Match (SchemaMatch)

The Schema level Match is the measure of structural similarity between two concepts. Many times concepts from both XML schema and ontologies are expressed in terms of other concepts. Hence while matching such concepts, it is important to match the sub-concepts tree under that concept also. SchemaMatch accounts for this by calculating the geometric mean of Sub-concept Similarity (subConceptSim) and the Sub-concept Match (subConceptMatch). Equation 3 gives the formula for SchemaMatch.

\[
\text{SchemaMatch} = \sqrt{\text{subConceptSim} \times \text{subConceptMatch}}
\]

where, \(\text{subConceptSim} \in [0,1]\) \(\text{subConceptMatch} \in [0,1]\)

### Equation 3. Formula for Calculating SchemaMatch

#### 3.2.2.1 Sub-concept Similarity (subConceptSim)

The Sub-concept Similarity \((\text{subConceptSim})\) is the average match score of each individual property of the concept.

\[
\text{subConcept Sim} = \frac{\sum_{i=1}^{n} \text{MS}(\text{subconcept}_i)}{n}
\]

where, \(n = \text{no of subconcept s of the main Concept}\)

### Equation 4. Formula for Calculating subConceptSim
3.2.1.2 Sub-concept Match (subConceptMatch):  
subConceptMatch (Equation 5) can be defined as the fraction of the total number of properties of a concept that are matched.

\[
\text{subConceptMatch} = \frac{n(\text{matched subConcept s})}{n(\text{total subConcept s})}
\]

Equation 5. Formula for Calculating subConceptMatch

Table 6 below shows how subConceptSim and subConceptMatch are calculated. Pressure is the WSDL concept with sub-concepts Delta, slp and relative_humidity and PressureEvent is the ontological sub-concept with properties Sea Level Pressure, RelativeHumidity etc.

<table>
<thead>
<tr>
<th>WSDL Concept</th>
<th>Ontological Concept</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>Slp</td>
<td>Sea Level Pressure</td>
<td>1</td>
</tr>
<tr>
<td>relative_humidity</td>
<td>RelativeHumidity</td>
<td>1</td>
</tr>
</tbody>
</table>

subConceptSim (Pressure, PressureEvent) = \((1+1+0)/3 = 0.667\)
subConceptMatch (Pressure, PressureEvent) = \(2/3 = 0.667\)

3.3 Finding the best mapping

As each WSDL concept is compared against all the concepts from ontologies, it is necessary to find the best matching concept. We have implemented a function getBestMapping listed in Table 7 for the same.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>getBestMapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTS</td>
<td>wc, w, o = {oc_1, oc_2, \ldots, oc_n}</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Best(m_i = (wc, oc, MS))</td>
</tr>
</tbody>
</table>

This algorithm maintains a variable for best mapping, whose MS is checked against the newly generated mapping. If the new mapping has a better MS, it is assigned as the best mapping. Since we are trying to find a match for a WSDL concept, while comparing with the ontological concept we only consider the number of children of the WSDL concept. This gives the same schema level match for the best matching ontological concept and its super-concepts. Therefore, it is necessary to implement some technique to rank the best matching ontological concept higher than its super-concepts. The getBestMapping function achieves this by considering the total number of sub-concepts of the two concepts being mapped.

For example, consider a WSDL concept PhenomenonType which best matches to the ontological concept CurrentWeatherPhenomena and WeatherPhenomena is the super-concept of CurrentWeatherPhenomena. From Table 8 we can see that both have the same SchemaMatch but WeatherPhenomena has a better ElementMatch making MS for it slightly better than the MS of CurrentWeatherPhenomena. Thus if we do ranking based on MS, WeatherPhenomena will get ranked higher. This can be avoided by considering the number of sub-concepts of both of them. From Table 4 we know that ElemMatch has very little weightage (0.2) if SchemaMatch is above 0.75. Also if we have two candidate concepts with same SchemaMatch value, then the concept with less number of sub-concepts is a better match. Thus ranking algorithm gives more weight to number of concepts than ElemMatch when SchemaMatch is same. Hence we are able to rank CurrentWeatherPhenomena higher than WeatherPhenomena.

3.4 Categorizing and annotating WSDL

Each Web service description i.e. the WSDL file is compared against all the ontologies in the Ontology-store (Explained in section 4). For every ontology, a set of mapping is created. Two measures are derived from these set of mappings; the first is the Average Concept Match (avgConceptMatch) and the second is the Average Service Match (avgServiceMatch).

3.4.1 Average Concept Match (avgConceptMatch)

The Average concept match tells the user about the degree of similarity between matched concepts of the WSDL schema and ontology. This measure is used to decide if the computed mappings should be accepted for annotation. It is normalized on the scale of 0 to 1 where 0 denotes no similarity and 1 denotes complete similarity. Equation 6 gives the formula for avgConceptMatch.

\[
\text{avgConceptMatch} = \frac{\sum_{i=1}^{k} \text{MS}(m_i)}{k}
\]

where \( k = \text{no of mapped concepts} \)

Equation 6. Formula for Calculating avgConceptMatch

3.4.2 Average Service Match (avgServiceMatch)

The Average service match helps us to categorize the service into categories. It is calculated as the average match of all the concepts of a WSDL schema and a domain ontology. The domain of the ontology corresponding to the best average service match also represents the domain of the Web service. The Average service match as shown in Equation 7 is also normalized on the scale of 0 to 1.

\[
\text{avgServiceMatch} = \frac{\sum_{i=1}^{k} \text{MS}(m_i)}{n}
\]

where \( k = \text{no of mapped concepts} \)  
\( n = \text{no of concepts WSDL Schema} \)

Equation 7. Formula for Calculating avgServiceMatch

We explain both these measures further with the example given in Table 9. From the table we can see that AirportWeather service matches better with Weather-ont ontology (5 out of 8 concepts
Therefore, the domain of AirportWeather service is Weather. Similarly, IMapQuest service is from Geographical domain.

### Table 9. avgServiceMatch and avgConceptMatch

<table>
<thead>
<tr>
<th>Web service</th>
<th>Ontology</th>
<th>Num concepts</th>
<th>avg Concept Match</th>
<th>avg Service Match</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>total</td>
<td>mapped</td>
<td></td>
</tr>
<tr>
<td>AirportWeather</td>
<td>Weather -ont</td>
<td>8</td>
<td>5</td>
<td>0.756</td>
</tr>
<tr>
<td>AirportWeather</td>
<td>Geo</td>
<td>8</td>
<td>2</td>
<td>0.655</td>
</tr>
<tr>
<td>IMapQuest</td>
<td>Geo</td>
<td>9</td>
<td>6</td>
<td>0.9</td>
</tr>
<tr>
<td>IMapQuest</td>
<td>Weather -ont</td>
<td>9</td>
<td>2</td>
<td>0.388</td>
</tr>
</tbody>
</table>

4. ARCHITECTURE

In this section we explain the architecture of the system. The three main components of the system are an ontology store, the matcher library, and a translator library.

4.1 Ontology-Store

Ontology-store as the name suggests stores the ontologies. These ontologies will be used by the system to annotate the Web service descriptions in WSDL. The ontologies are categorized into domains. The system allows the user to add new ontologies to the ontology store. Currently the system supports DAML, and RDF-S ontologies. These ontologies are stored as “.daml” or “.rdfs” files in different folders. Names of these folders correspond to domain names. This component of our architecture will be replaced by a high quality search mechanism of ontologies from ontology registries or a P2P mechanism supporting semantic search of ontologies [8].

4.2 Translator Library

The translator library consists of the programs that are used to generate the SchemaGraph representations (explained in section 3.1). Currently, the translator library provides two translators, WSDL2graph and Ontology2graph. WSDL2graph takes as input the WSDL file to be annotated and generates the SchemaGraph representation, which is fed to the matching algorithm. In a similar manner the Ontology2Graph generates the SchemaGraph for the ontology.

4.3 Matcher Library

The matcher library provides two types of matching algorithms, element level matching algorithms and schema matching algorithms.

Currently only one schema matching algorithm, findGraphMatch is implemented. Element level matching algorithms provided by the library include NGram, TokenMatcher, CheckSynonyms and CheckAbbreviations which are detailed in section 3.2.1. The Matcher library also provides user with option to add new matching algorithms using an API. Figure 2 shows the interface for selecting existing element level algorithms and for adding new ones.

5. RESULTS AND EMPIRICAL TESTING

To test our algorithm we first obtained a corpus of 424 Web services from SALCentral.org and XMethods.com. Although our initial intention was to test our algorithm on the whole corpus, we have limited our testing to two domains, due to lack of relevant domain specific ontologies. We are in the process of creating new domain ontologies and plan to extend our testing for remaining Web services in the future.

The two domains we have selected for testing are Weather and Geographical domains. Although the ontologies used are not comprehensive enough to cover all the concepts in these domains, they are sufficient enough to serve the purpose of categorization. We have taken a set of 24 services out of which 15 are from geographical domain and 9 from weather domain. The services

2 Acknowledgement: Andreas Hess and N. Kushmeric [18] for lending us the corpus
are categorized based on the categorization threshold (CT), which decides if the service belongs to a domain. If the best average service match (section 3.4.2) calculated for a particular Web service is above the CT then the service belongs to the corresponding domain. Graph 1 depicts the categorization obtained by applying our algorithm on this set of 24 Web services for different CT values.

Graph 1. Categorization statistics of Web services

It is very important to choose the CT value correctly. We can see from Graph 1, that for CT = 0.5, very few services have been categorized. Whereas for CT = 0.4, although all Web services are categorized, two services from the weather domain have been wrongly categorized in the geographical domain. These two services are WorldWeather and ForecastByICAO. Both these services take “ICAO code” as input and return the “weather as an array of string”. As the output is not described in terms of concepts from weather domain and the categorization is based only on the input concept “ICAO code” (which is mapped to concept from Geo ontology), these services are wrongly categorized.

Graph 2. Selecting domain for a Web service

Graph 2 summarizes the categorization process of 6 different Web services. These services are compared to 5 different ontologies and the average service match scores are obtained. A service belongs to the category of the domain ontology for which it gives the best match score. For example, the second service in the graph i.e. the AirportWeather service best matches to the “Weather-ont” ontology and hence belongs to the weather domain. The match scores for other domain ontologies suggest that, this service may contain a few concepts from these other domains.

Graph 3 shows two plots of match scores of 17 Web services (categorized in geographical domain) compared with two versions of domain specific Geo ontology. The lower plot shows Match Scores with the original Geo ontology. We can see that the Match Scores are quite low because the Geo ontology (number of concepts = 94) is not comprehensive enough to contain all the concepts from the geographical domain. This observation is proved by the upper plot, which shows a significant increase in Match Scores of these Web services, when compared with the new Geo ontology with a few added concepts.

Graph 3. Mappings with Geo ontology – Match Scores

Graph 4 gives a comparison between total number of concepts and the number of mapped concepts for all the 17 Web services. The topmost plot shows total number of concepts in web services, the plot at bottom shows number of mapped concepts before adding new concepts to Geo ontology and the middle plot shows the number of concepts mapped after adding new concepts to the ontology. This plot also supports the fact that matches are low due to the incomplete domain ontology.

Graph 4. Mappings with Geo ontology – Number of concepts
Although Graph 3 and Graph 4 show that low match scores can be improved with better ontologies, still we can see that some of the Web services do not show much increase in the Match Scores. The reason behind this is many Web services span more than one domain and hence contains concepts from domains other than the geographical domain. Also as WSDL files are generated automatically by web servers, the input and output parameters do not always have meaningful names.

6. RELATED WORK

Our work presents an approach for adding semantics to Web services. In this section, we discuss some other efforts that describe adding semantics to Web services. We also look into some schema matching efforts, as it is the basis of our approach to semantically describe Web services.

DAML-S (soon to be OWL-S) uses an upper ontology to semantically describe Web services. We share the vision of adding semantics to Web services by using annotated WSDL descriptions in our previous work [8]. The common factor in the aforementioned two efforts is in mapping the message parts in WSDL to ontologies. With the potential growth in Web services, finding relevant ontologies for a particular service will be a significant problem. An even more difficult task will be to map the concepts in the ontologies to elements in WSDL. Even though DAML-S assumes manual annotation of Web services, we believe that annotation in the real world will be a non-trivial task, without some degree of automation. This work primarily aims on providing a semi-automatic approach to matching elements in WSDL to ontologies. [18] talks about using semantic metadata to semi-automatically categorize Web services into predefined categories making the service discovery simpler. It uses machine learning techniques for categorization. There are two significant differences in our approach and that suggested in [18]. First, we believe our approach is richer as we consider the structure of WSDL concepts, rather than just the names. Secondly, we use ontologies for classification as compared to vocabularies used by [18]. Ontologies are more descriptive and capture domains more accurately than vocabularies, leading to better classification.

Since we are matching XML schema used by the WSDL files to ontologies, it is worthwhile to explore the Ontology matching and Schema matching areas. Mapping ontologies is a hard problem [19]. The research in this area varies from ontology merging [20] to mapping ontologies for service discovery [21]. The techniques used are also varied, ranging from machine learning [22][23], graph analysis [20][24] to heuristic based matching [24]. Schema matching is an old research area and there has been a lot of research in this area from different perspectives [24], which is also related to earlier schema integration work [26][27][28]. There are different approaches to schema matching like matching the whole schema structure versus matching the individual elements of the schema. There are many machine learning techniques [29][30][31] where some matching rules are fed to the match algorithm and then it guesses the new matches. Some match algorithms use more than one technique and called hybrid matchers. Due to space limitation, we are not able to discuss all of them in this paper. Rather, we focus on two of the more relevant schema matching techniques and their relationship to our work, namely COMA [32] and Cupid [33].

Cupid is a hybrid matcher which combines name matching with structure matching. It uses predefined synonym dictionary to find element level matches. Every schema node has two dimensions of similarity; the element level match calculated using name matches and predefined synonym dictionary and structure match. COMA implements a matcher library which has different matchers varying from simple matchers like name, soundex, and synonym matchers to hybrid matchers using name and path information. Although these matching techniques are different and find the matches using different algorithms, some of the basic steps like name matching, tokenization, word expansion, finding words with similar meaning etc. are common. In fact even though the implementations are different, these steps are the basis of the most of the schema matching techniques.

In this paper, we have discussed annotation of input and output concepts of Web services. Relating Web services to process ontologies has been discussed in [34]. We are currently working on algorithms to map operations in WSDL files to concepts in process ontologies.

7. CONCLUSION AND FUTURE WORK

In this paper we have described MWSAF, a framework for semi-automated annotation of Web services. We have discussed the issues in matching XML schemas to ontologies, which forms the crux of our approach. This work was undertaken as a part of the METEOR-S system. While many other efforts have talked about adding semantics to Web services, practical implications of actually annotating Web services with real world ontologies have not been discussed in great detail. We further carried out experiments involving Web services and ontologies independently created by others, and coped with the practical difficulty in our effort due to lack of domain ontologies and well structured WSDL files. This prototyping and early experimentation leads us to believe that our approach will scale well when the users will have to deal with thousands of Web services, but also have the benefit of higher quality and more comprehensive ontologies. We plan to release our tool for public use through sourceforge. We are currently working on completing the documentation and user guide for this public release.

8. REFERENCES


9. APPENDIX A

Figure 4. Screenshot of the MW SAF tool

Figure 4 above gives a screenshot of the MW SAF tool. The user first loads the WSDL file (1) to be mapped. This WSDL file is compared with all the ontologies from the ontology-store to find the most suitable domain ontology using the “findDomain” option from the “Tools” menu. This option returns the match scores with each ontology (2). The best-matched ontology can then be selected for annotation. Mappings for this ontology can be viewed and the user can accept or reject suggested mappings (3). The tool also allows viewing of mappings with other ontologies, in case if the WSDL file contains concepts from other domains. There is also a facility to add extra mappings manually. The WSDL file and ontology can be viewed in a tree format (1) and (5) respectively to facilitate manual mapping. (4) shows accepted mappings, which are then written to the WSDL file as shown in Figure 5.

```
+xml:complexType Ont-Concept="weather:windDirection" name="Direction">
  <xsd:complexType name="Station">
    <xsd:sequence>
      <xsd:element Ont-Concept="geo:icao" maxOccurs="1" minOccurs="1" name="icao" nillable="true" type="xsd:string"/>
      <xsd:element Ont-Concept="geo:wmo" maxOccurs="1" minOccurs="1" name="wmo" nillable="true" type="xsd:string"/>
      <xsd:element Ont-Concept="geo:iata" maxOccurs="1" minOccurs="1" name="iata" nillable="true" type="xsd:string"/>
      <xsd:element Ont-Concept="geo:elevation" maxOccurs="1" minOccurs="1" name="elevation" type="xsd:double"/>
      <xsd:element Ont-Concept="geo:latitude" maxOccurs="1" minOccurs="1" name="latitude" type="xsd:double"/>
      <xsd:element Ont-Concept="geo:longitude" maxOccurs="1" minOccurs="1" name="longitude" type="xsd:double"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:complexType>
```