METEOR-S WEB SERVICE ANNOTATION FRAMEWORK

by

ABHIJIT PATIL

(Under the Direction of Amit Sheth)

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. Although current standards like UDDI, WSDL, and SOAP form the basis of making Web services a workable and broadly adopted technology they are syntactic and therefore are plagued with problems like inefficient service discovery and composition. Semantic Web technologies are likely to provide better qualitative and scalable solutions to these problems. Semantic annotation of Web services is a critical first step to achieve the above promise. In this work 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 and to categorize Web services into domains. An empirical study of our approach is presented to help evaluate their performance.

INDEX WORDS: Semantic Web services, WSDL, Ontology, semantic annotation of Web services, Web services discovery, METEOR-S

METEOR-S WEB SERVICE ANNOTATION FRAMEWORK

by

ABHIJIT PATIL

B.E., University of Mumbai, 2000

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

2003

METEOR-S WEB SERVICE ANNOTATION FRAMEWORK

by

ABHIJIT PATIL

Major Professor:	Amit Sheth
Committee:	John A. Miller
	I. Budak Arpinar

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2003

DEDICATION

To my mother Vinaya

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Amit P. Sheth for his guidance, foresight, and his assurance in difficult times. Dr. Sheth has been very generous with his time and wisdom. I would like to thank my parents and brother for their continuous encouragement and love. I would also like to thank Dr. John A. Miller and Dr. I. Budak Arpinar for their valuable suggestions and being a part of my committee. Special thanks to my friends Swapna Oundhakar, Maria Chinwala and Kunal Verma for their support and encouragement.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

The World Wide Web has grown exponentially and is steadily evolving from being just a web of data to a web of services. Web services based on current industry standards like UDDI, WSDL and SOAP promise to deliver a comprehensive solution for businesses by creating a platform for seamless e-commerce. These standards, however, are in their fledgling years and focus only on operational and syntactic details for implementation and execution. The search mechanism therefore becomes restricted to keyword-based searches.

Research in the semantic web area has pointed out that annotation with metadata can help us solve the problem of inefficient keyword based searches in the current web based on HTML, HTTP, and URI. This concept of annotation can be extended to web services to envision semantic web services. Semantically described services will enable better service discovery and also allow easier interoperation among services.

There have been a number of approaches suggested for adding semantics to Web services. Semantics can either be added to existing web service standards like UDDI or WSDL [1] or they can explicitly embeded in the description itself [2]. The common factor in most approaches is relating concepts in Web services to domain specific ontologies. We refer to relating concepts to ontologies as annotation. While significant research has been done on what to annotate, there has been little work on how to annotate.

Current Industry efforts like UDDI, WSDL, and SOAP are in their fledgling years and have been going on in earnest to make web services a workable technology. Our approach is to work around these existing 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. MWSAF helps in describing services semantically and aids efficient web service discovery and composition.

CHAPTER 2

METEOR-S WEB SERVICE ANNOTATION FRAMEWORK (MWSAF)[1]

[1] Abhijit Patil, Swapna Oundhakar, Amit Sheth, Kunal Verma. Submitted to *Thirteenth International World Wide Web Conference*

1. INTRODUCTION

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 URIs). 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 average real 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 are 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 work. 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][8].

- 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 [9]. 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. METEOR-S Web Service Discovery Infrastructure (MWSDI) provides an infrastructure to leverage data, functional and QoS semantics by enhancing UDDI [10]. The top layer, i.e., the flow layer deals mainly with service composition. A comprehensive framework for Semantic Web service composition is provided in METEOR-S Web Service Composition Framework (MWSCF) [11]. 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.

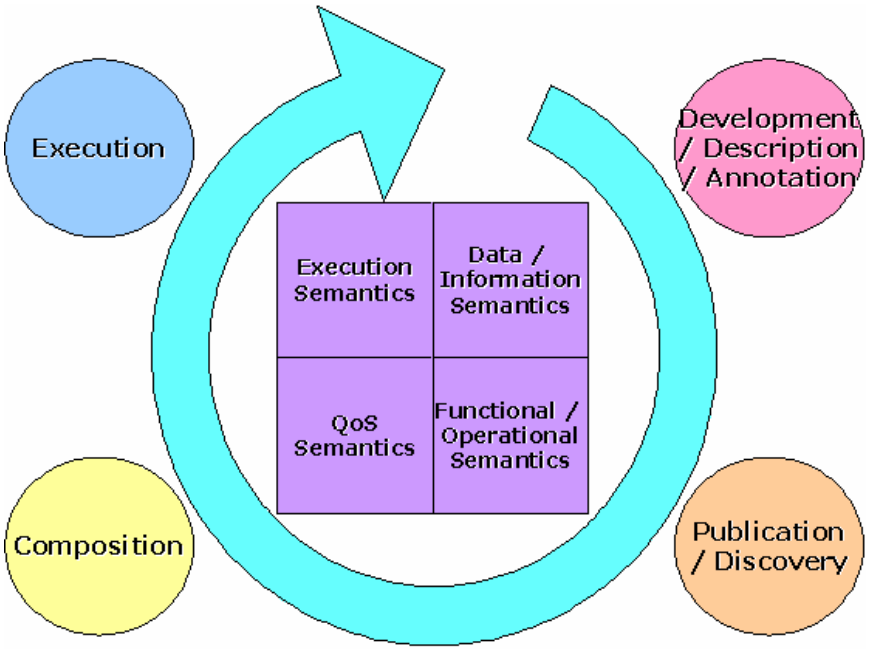


Figure 1. Four types of Semantics in Web services

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 [12]. 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 [13]. 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. A SchemaGraph is a set of nodes connected by edges. We use conversion functions to convert both XML schema and ontology[2] to SchemaGraphs.

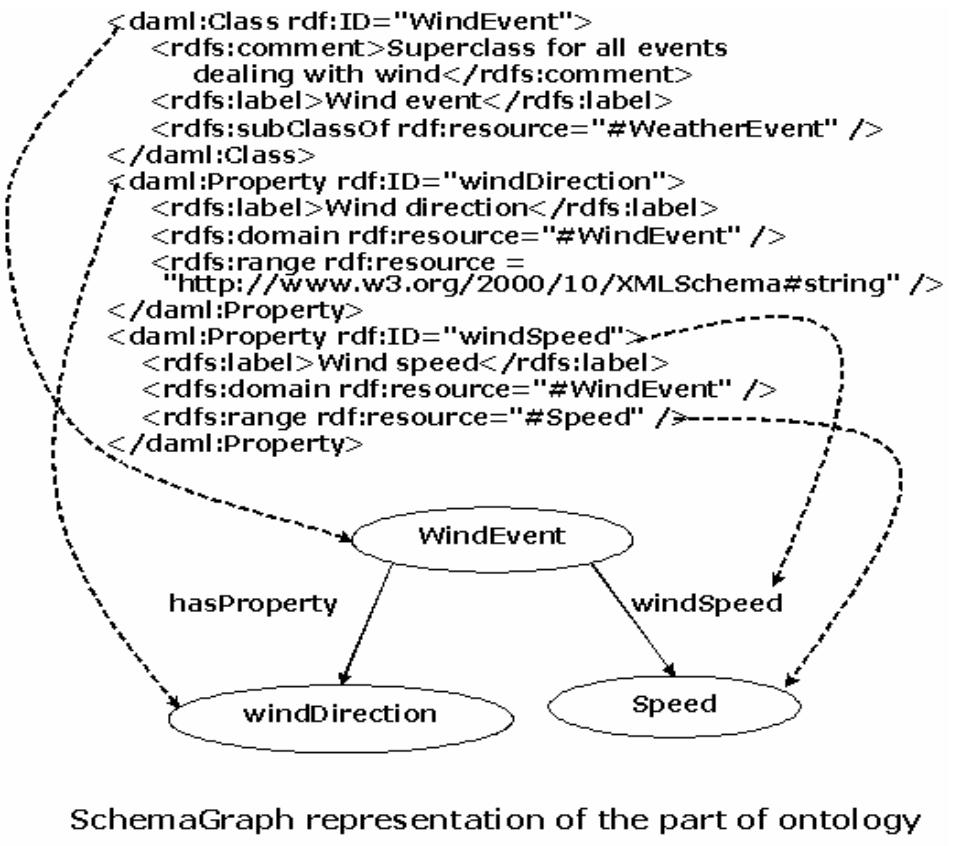
The conversion function used for converting Ontology to SchemaGraph is *Ontology2Schema* and uses the conversion rules specified in Table 1.

[2] Currently we use ontologies represented using RDF-S, subset of DAML+OIL and OWL

Table 1. Ontology to SchemaGraph Conversion Rules

Ontology representation	**SchemaGraph representation**
Class	Node
Property with basic datatypes as range (Attribute)	Node with edge joining it to the class with name “hasProperty”
Property with other class as range (Attribute)	Edge between the two class nodes
Instance	Node with edge joining it to the class with name “hasInstance”
Class – subclass relationship	Edge between class node to subclass node with name “hasSubClass”

Example



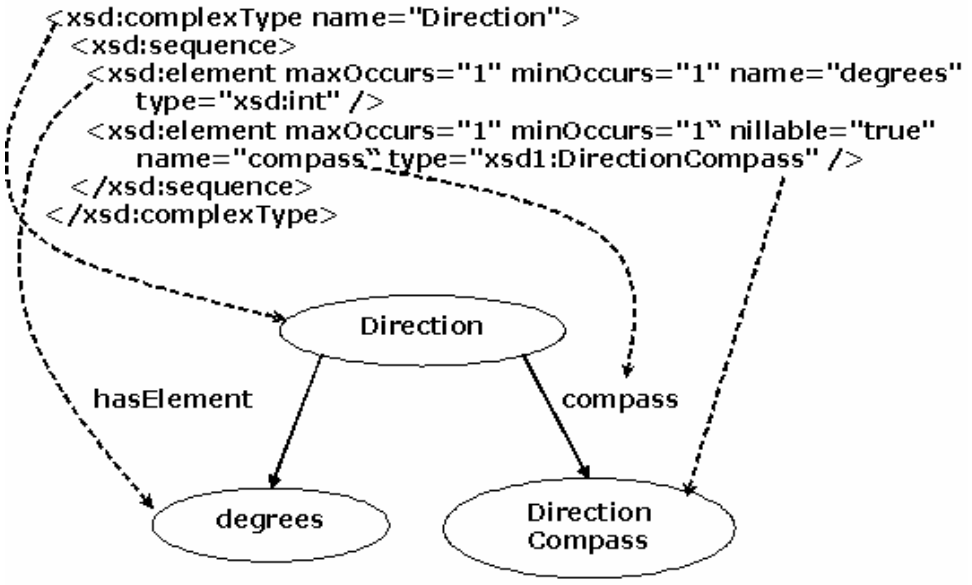
SchemaGraph representation of the part of ontology

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

Table 2. XML Schema to SchemaGraph Conversion Rules

XML schema Construct	SchemaGraph representation
ComplexType	Node
Elementary XML Data Type Element defined under complexType	Node and an Edge between complexType node and this node with name "hasElement"
ComplexType XML Data Type Element defined under complexType	Edge
SimpleType	Node
Values defined for simple types	Node and edge between simpleType and this node with name "hasValue"
Elements	Nodes

Example



SchemaGraph representation of the part of WSDL

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 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_i, i.e., WSDL concept, oc_i, i.e., ontology concept and MS, i.e., Match Score.

Table 3. Overview of Function findMapping

FUNCTION	findMapping
INPUTS	$wc_i \in W$, $oc_i \in O$ where, W is the set of all elements in a WSDL file, W = {wc_1, wc_2, wc_3, …., wc_n} in SchemaGraph representation and O is the set of ontological concepts of an Ontology denoted by O = {oc_1, oc_2, oc_3, …, oc_m} in SchemaGraph representation
OUTPUT	$m_i = (wc_i, oc_j, MS)$ where, m_i is the mapping between wc_i and oc_j and MS is the Match Score calculated for the mapping wc_i and oc_j ($MS \in [0,1]$)

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.

$$MS = \frac{w1 * ElemMatch + w2 * SchemaMatch}{w1 + w2}$$
$$\text{where,} \quad (0 \le w1 \le 1) \quad (0 \le w2 \le 1)$$

Equation 1. Formula for Calculating Match Score (MS)

Weights w1 and w2 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. Also as WSDL documents are generated automatically by web

servers, not always they have meaningful names to the concepts. Hence we give more weightage to schema level match than element level matchBased on these conditions the values of w1 and w2 are changed as shown in Table 4.

Table 4. Weight values for Calculating MS

Condition	**w1**	**w2**
Default	0.4	0.6
WSDL concept is leaf node	1	0
SchemaMatch > 0.9, ElemMatch < 0.9	0.1	0.9
SchemaMatch > 0.75, ElemMatch < 0.75	0.2	0.8
SchemaMatch > 0.65, ElemMatch < 0.65	0.3	0.7
SchemaMatch < 0.5, ElemMatch > 0.5 and WSDL Concept is of SimpleType	1	0
SchemaMatch < 0.5, ElemMatch < 0.5 and WSDL Concept is of SimpleType	0.5	0.5

3.2.1. Element level Match (ElemMatch)

The Element level match (*ElemMatch*) is the measure of the linguistic similarity between two concepts based on their names. Here we assume that the concepts from WSDL and ontologies have meaningful names. The ElemMatch function uses various name and string matching algorithms like NGram, synonym matching, abbreviation expansion, stemming, tokenization etc. The *NGram* algorithm calculates the similarity by considering the number of qgrams [14][15][16] that the names of two concepts have in common. The *CheckSynonym* algorithm uses WordNet [17] to find synonyms whereas, a custom abbreviation dictionary is used by the *CheckAbbreviations* algorithm. The *TokenMatcher* uses the Porter Stemmer [18] algorithm, tokenization, stop-words removal, and substring matching techniques to find the similarity. It first tokenizes the string based on punctuation and capitalization. Then it removes unnecessary words from the list of tokens, using a stop-word list. If it can not match these individual token then it stems them using porter stemmer algorithm and tries to match them using NGram technique. If any of these algorithms return a full match, i.e., 1 on scale of 0 to 1, then a match score of 1 for linguistic similarity is returned. If all the match algorithms give a match value of zero, then the linguistic

similarity of those concepts is 0. If on the other hand, none of the match algorithms give a match score of 1, i.e., an exact match, then the average of all non-zero match scores is taken. Equation 2 and Table 5 explains all these cases with examples.

$$\text{ElemMatch} = \begin{cases} 1 & \text{if}(ms1 \vee ms2 \vee ms3 = 1) \\ ms1 & \text{if}((0 < ms1 < 1) \wedge (ms2 = ms3 = 0)) \\ 0 & \text{if}(ms1 = ms2 = ms3 = 0) \end{cases}$$

$$\text{where,} \quad ms1 = \text{MatchScore}(\text{NGram})$$
$$ms2 = \text{MatchScore}(\text{SynonymMatching})$$
$$ms3 = \text{MatchScore}(\text{AbbreviationExpansion})$$

Equation 2. **Formaula for Calculating ElemMatch**

Table 5. Examples of ElemMatch

WSDL Concept	Ontological Concept	ElemMatch	Algorithm
wind	WindEvent	0.639	NGram
wind	WindChill	0.478	NGram
snow	Snowfall	1	Synonyms
slp	Sea Level Pressure	1	Abbreviation
relative_humidity	RelativeHumidity	1	NGram

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.

$$\text{SchemaMatch} = \sqrt{\text{subConceptSim} * \text{subConceptMatch}}$$
$$\text{where,} \text{subConceptSim} \in [0,1] \quad \text{subConceptMatch} \in [0,1]$$

Equation 3. **Formula for Calculating SchemaMatch**

3.2.1.1 Sub-concept Similarity (subConceptSim)

The Sub-concept Similarity (*subConceptSim*)(Equation 4) is the average match score of each individual property of the concept.

$$\text{subConceptSim} = \frac{\sum_{i=1}^{n} \text{MS}(\text{subconcept}_i)}{n}$$

$$\text{where}, n = \text{no of subconcepts of the main Concept}$$

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.

$$\text{subConceptMatch} = \frac{n(\text{matched subConcepts})}{n(\text{total subConcepts})}$$

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 6. Calculations of SchemaMatch

WSDL Concept Pressure	**Ontological Concept PressureEvent**	**MS**
Delta	----	0
Slp	Sea Level Pressure	1
relative_humidity	RelativeHumidity	1
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 7. Overview of Function getBestMapping

FUNCTION	getBestMapping
INPUTS	$wc_i \in W$, $O = \{oc_1, oc_2, oc_3, \ldots, oc_m\}$
OUTPUT	$Best(m_i = (wc_i, oc_j, MS))$

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.

Table 8. Mappings for WSDL Concept PhenomenonType

OntologyConcept	ElemMatch	SchemaMatch	No. subconcepts	MS	Rank
Weather Phenomena	0.614	0.854	106	0.81	2
OtherWeather Phenomena	0.442	0.396	13	0.42	3
CurrentWeather Phenomena	0.564	0.854	35	0.79	1

For example, consider a WSDL concept PhenomenonType which ideally 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 weight (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 the 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*.

$$\text{avgConcept Match} = \frac{\sum_{i=1}^{k} \text{MS}(m_i)}{k}$$
$$\text{where}, \quad k = \text{no of mapped concepts}$$

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. More the number of matched concepts with a domain ontology, more the probability of service belonging to that domain. Thus

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 normalized on the scale of 0 to 1.

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

$$\text{where, } \begin{array}{l} k = \text{no of mapped concepts} \\ n = \text{no of concepts WSDL Schema} \end{array}$$

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 ot of 8 concepts mapped) than *Geo* ontology (2 out of 8 concepts mapped). Therefore, the domain of *AirportWeather* service is *Weather*. Similarly, *IMapQuest* service is from *Geographical* domain.

Table 9. Calculations of avgServiceMatch and avgConceptMatch

Web service	Ontology	Num concepts		AvgConcept Match	AvgService Match
		Total	Mapped		
AirportWeather	Weather-ont	8	5	0.756	0.47
AirportWeather	Geo	8	2	0.655	0.16
IMapQuest	Geo	9	6	0.9	0.6
IMapQuest	Weather-ont	9	2	0.388	0.075

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 parser 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, OWL and RDF-S ontologies. These ontologies are stored as ".daml" or ".rdfs" or ".owl" 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 [9].

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.

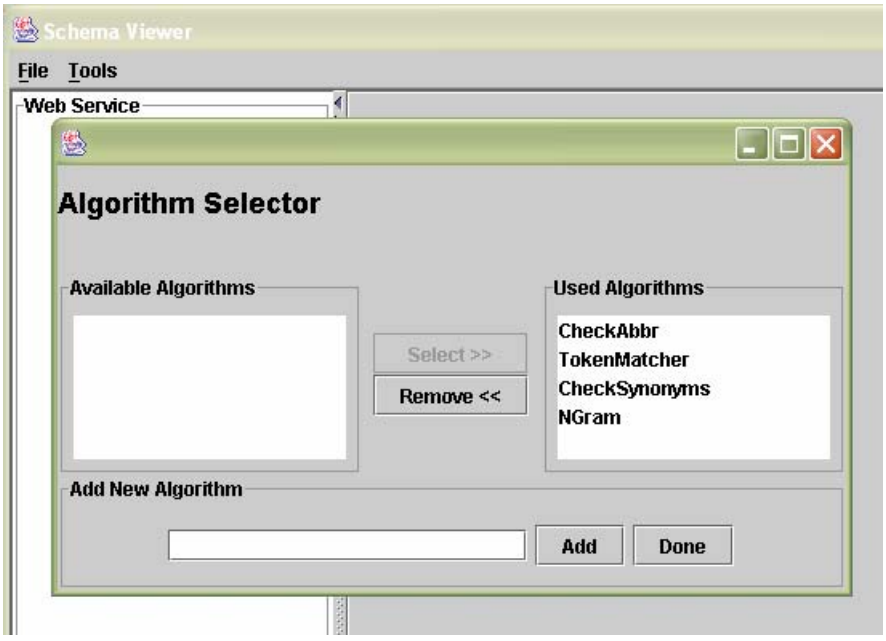


Figure 2. Matcher Library – Algorithm Selector

Currently only one schema matching algorithm, *findGraphMatch* is implemented. Element level matching algorithms provided by the library include *NGram*, *TokenMatcher*, *CheckSysnonyms* 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.

Once the *getBestMapping* function returns a set of best mappings for the WSDL schema the mappings can be displayed using the user interface. The user is provided with the ability to accept or reject the suggested mappings. Concepts can also be matched manually. The user can also visualize the WSDL descriptions and ontologies in a tree format. Once the mappings are accepted, they are written back to the WSDL file (Appendix A – Figure 5). Figure 3 gives an overview of MWSAF architecture.

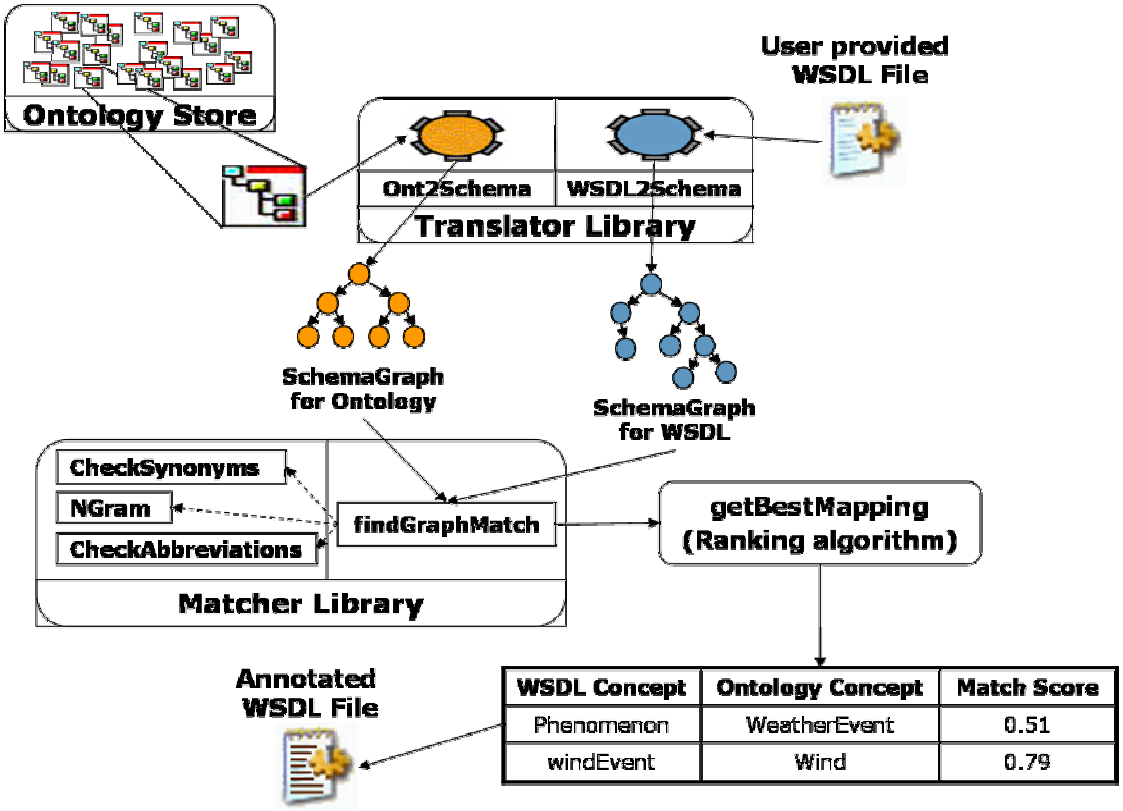


Figure 3. MWSAF – Architecture

5. RESULTS AND EMPIRICAL TESTING

To test our algorithm we first obtained a corpus[3] 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 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. Figure 4 depicts the categorization obtained by applying our algorithm on this set of 24 Web services for different CT values.

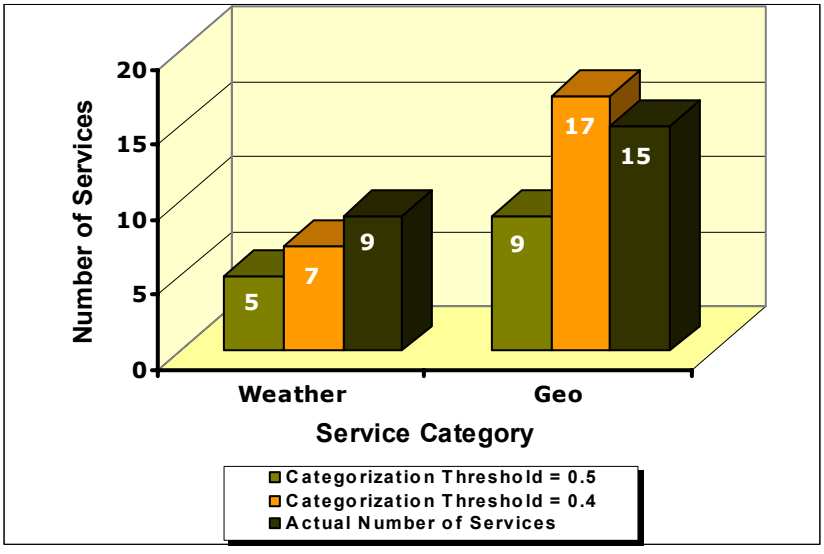


Figure 4. Categorization Statistics of Web services

[3] Acknowledgement: Andreas Hess and N. Kushmeric [19] for lending us the corpus

It is very important to choose the CT value correctly. We can see from Figure 4, 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.

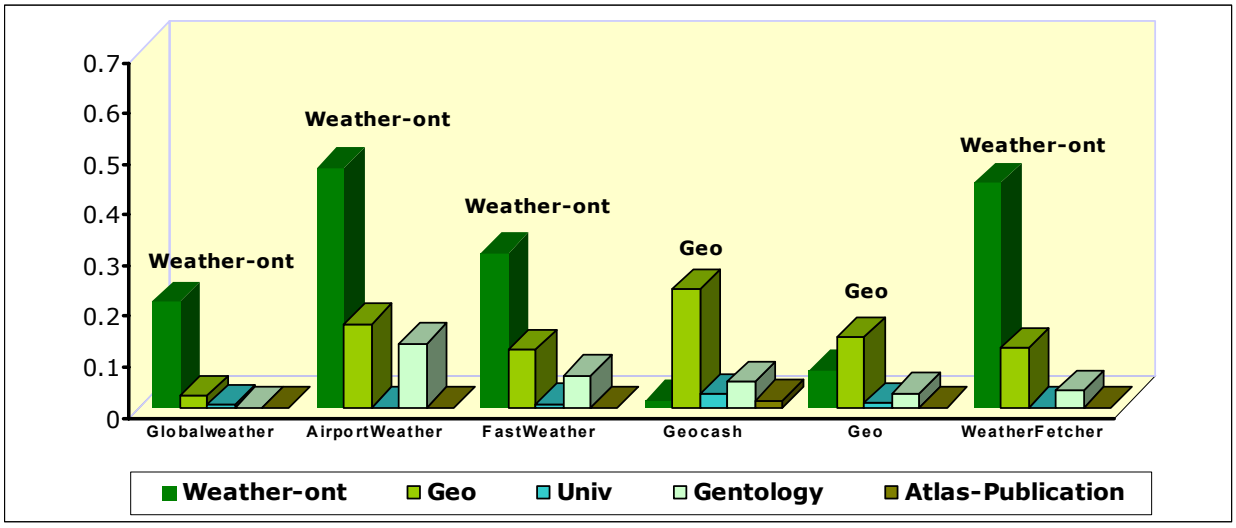


Figure 5. Selecting Domain for a Web service

Figure 5 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.

Figure 6 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.

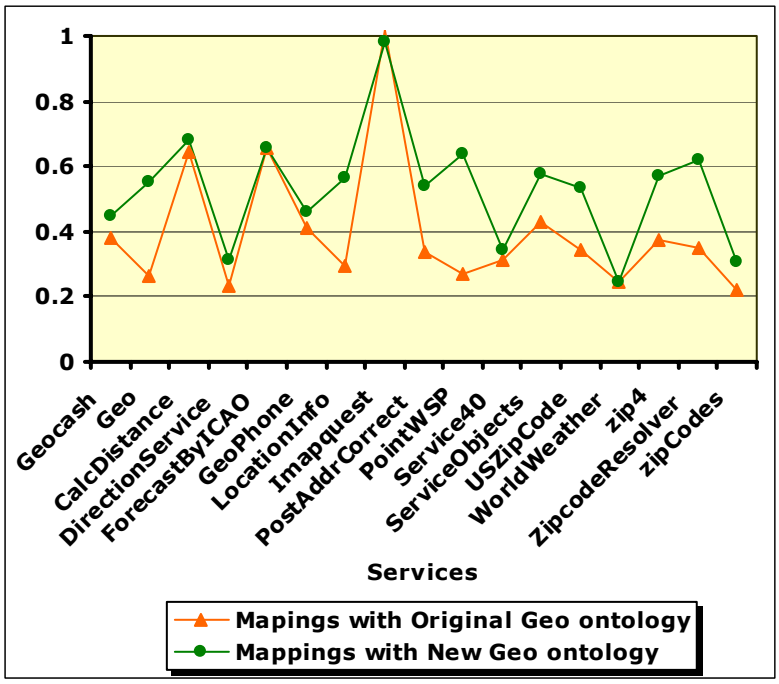


Figure 6. Mappings with Geo Ontology – Match Scores

Figure 7 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.

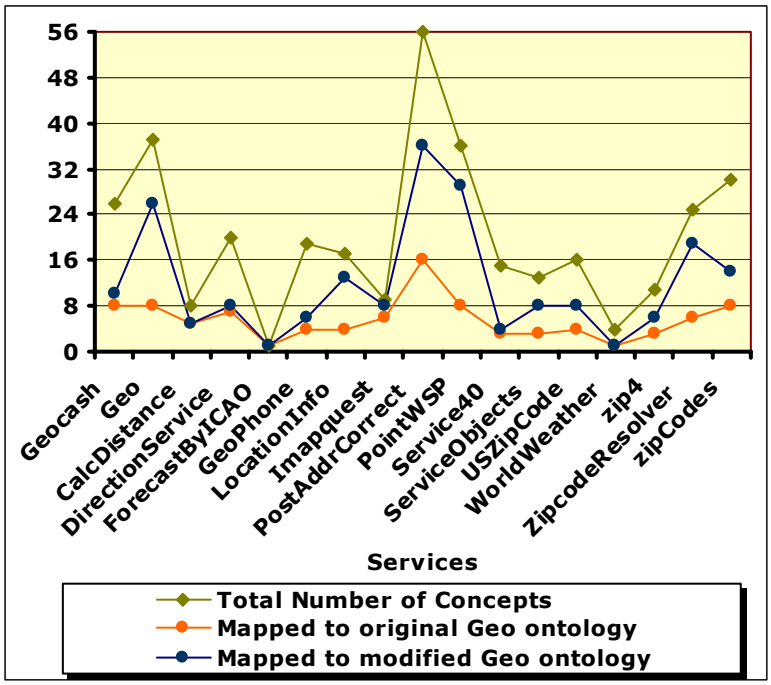


Figure 7. Mappings with Geo Ontology – Number of Concepts

Although Figure 6 and Figure 7 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 [9]. 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. [19] 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 [19]. 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 [19]. 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 [20]. The research in this area varies from ontology merging [21] to mapping ontologies for service discovery [22] and automatic service composition [23]. The techniques used are also varied, ranging from machine learning [24][25], graph analysis [21][26] to heuristic based matching [26]. Schema matching is an old research area and there has been a great deal of research in this area from different perspectives [26], which is also related to earlier schema integration work [27][28][29]. 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 [30][31][32] 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 [33] and Cupid [34].

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 [35]. 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-automatic 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.

CHAPTER 3

CONCLUSIONS

This work describes, MWSAF, a part of the METEOR-S project that provides a framework for semi-automatic annotation of Web services. We have discussed implementation of our prototype and issues in matching XML schemas to ontologies. Although many other efforts have talked about adding semantics to Web services, we have not seen prior efforts investigating practical implications of annotating Web services with real world ontologies. In our work, we faced practical challenges due to lack of existing well structured WSDL files and the relevant domain ontologies. We started with a collection of real world Web services in a domain. With improved matching algorithm and extending a relevant existing ontology to improve matching, we were able to complete a set of experiments. These helped us investigate our primary value proposition of the need for high quality semantic annotation for making Web services based environment and infrastructure more scalable. We plan to release our tool for public use shortly.

REFERENCES

[1] METEOR-S: Semantic Web Services and Processes, http://lsdis.cs.uga.edu/proj/meteor/SWP.htm

[2] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara, "DAML-S: Web service Description for the Semantic Web," *in Proceedings of the 1st International Semantic Web Conference* (ISWC 2002)

[3] S. Agarwal, S. Handschuh, and S. Staab "Surfing the Service Web", *in Proceedings of the 2nd International Semantic Web Conference* (ISWC 2003)

[4] Sheth A., Ramakrishnan C. "Semantic (Web) Technology In Action Ontology Driven Information Systems for Search, Integration and Analysis". *To appear in Data Engineering special issue on the Semantic Web.* December2003

[5] P. Holland, "Building Web Services From Existing Application". *eAI Journal*, September 2002, 45-47.

[6] D. Fensel, and C. Bussler, The Web service Modeling Framework. Vrije Universiteit Amsterdam (VU) and Oracle Corporation

[7] A. Sheth, "Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration," *Invited Talk, WWW 2003 Workshop on E-Services and the Semantic Web,* Budapest, Hungary, May 20, 2003.

[8] K. Sivashanmugam, A. Sheth, J. Miller, K. Verma, R. Aggarwal, P. Rajasekaran, Metadata and Semantics for Web Services and Processe, in Datenbanken und Informationssysteme: Festschrift zum 60- Geburtstag von Gunter Schlageter, Benn et al Eds., Praktische Informatik I, Hagen, October 2003, pp. 245-272.

[9] Web service Conceptual Architecture (WSCA 1.0), IBM Technical White Paper, May 2001, *http://www- 3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf*

[10] K. Verma, K. Sivashanmugam., A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web services", *Journal of Information Technology and Management* (to appear, 2004)

[11] K. Sivashanmugam, J. Miller, A. Sheth, and K. Verma, "Framework for Semantic Web Process Composition", Technical Report 03-008, LSDIS Lab, Computer Science Dept., UGA, http://lsdis.cs.uga.edu/lib/download/TR03-008.pdf

[12] M. Klein, D. Fensel, F. Harmelen, and I. Horrocks, "The relation between ontologies and XML Schemata", *Proceedings of the {ECAI}'00 workshop on applications of ontologies and problem-solving methods,* Berlin, August 2000

[13] D. Fensel, "Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce." SpringerVerlag, 2001.

[14] R. C. Angell, G. E. Freund, et al. "Automatic spelling correction using a trigram similarity measure." *Information Processing and Management* ***19****(4): 255-161*, 1983

[15] G. Salton, Automatic Text Processing: The Transformation, *Analysis and Retrieval of Information by Computer,* Massachusetts, Addison-Wesley, 1988

[16] E. Zamora, J. Pollock, et al. "The Use of Trigram Analysis for Spelling Error Detection." *Information Processing and Management* ***17****(6): 305-316,* 1981

[17] G. Miller. "Special Issue, WordNet: An on-line lexical database" International Journal of Lexicography, Vol. 3, Num. 4, 1990

[18] M. Porter, "An algorithm for Suffix Stripping", *Program – Automated Library and Information Systems*, 14(3):130-137, 1980

[19] A. Hess and N. Kushmerick, "Automatically attaching semantic metadata to Web services", *in Proceedings of the 2nd International Semantic Web Conference* (ISWC 2003)

[20] M. Klein, "Combining and relating ontologies: an analysis of problems and solutions". *in* (IJCAI 2001).

[21] N. Noy and M. Musen. "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment". *In Proceedings of the National Conference on Artificial Intelligence* (AAAI 2000).

[22] J. Cardoso and A. Sheth, Semantic e-Workflow Composition

[23] Ruoyan, Z., Arpinar, B., Aleman-Meza, B., Automatic Composition of Semantic Web Services, The 2003 International Conference on Web Services (ICWS'03), June 2003.

[24] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, Learning to Map between Ontologies on the Semantic Web, *Describes the GLUE system,* (WWW 2002)

[25] G. Stumme and A. Mädche. "FCA-Merge: Bottom-up merging of ontologies". *In 7th Intl. Conf. on Artificial Intelligence* (IJCAI '01), pages 225–230, Seattle, WA, 2001.

[26] P. Mitra, G. Wiederhold, and M. Kersten. "A graph-oriented model for articulation of ontology interdependencies". *In Proceedings Conference on Extending Database Technology 2000* (EDBT'2000), Konstanz, Germany, 2000.H. Do, S. Melnik, and E. Rahm. "Comparison of schema matching evaluations." *In Proceedings of the 2nd Int. Workshop on Web Databases (German Informatics Society)*, 2002.

[27] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. *In Proceedings of DOOD'93*, pages 81-100, Phoenix, AZ, December 1993.

[28] I. Schmitt and C. Türker, "An incremental approach to schema integration by refining extensional relationships", *Proceedings of the seventh international conference on Information and knowledge management*, 1998,

[29] F. Hakimpour and A. Geppert. “Resolving semantic heterogeneity in schema integration: An ontology based approach.” *In Chris Welty and Barry Smith, editors, Proceedings of International conference on Formal Ontologies in Information Systems FOIS'01.* ACM Press, October 2001.

[30] J. Berlin, and A. Motro: Autoplex: Automated Discovery of Content for Virtual Databases. *CoopIS* 2001, 108–122

[31] A. H. Doan, P. Domingos, and A. Halevy: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. *SIGMOD 2001*

[32] A. H. Doan, J. Madhavan, P. Domingos, and A. Halevy: Learning to Map between Ontologies on the Semantic Web. *WWW 2002*

[33] Hong-Hai Do and E. Rahm. “COMA - A System for Flexible Combination of Schema Matching Approaches”, *In Proceedings of the 28th International Conference on Very Large Databases* (VLDB), 2002.

[34] J. Madhavan, P. Bernstein, and E. Rahm. „Generic Schema Matching with Cupid”. *In Proceedings of the International Conference on Very Large Databases* (VLDB), 2001.

[35] M. Klein, and A. Bernstein. “Searching for Services on the Semantic Web using Process Ontologies.”, *in The First Semantic Web Working Symposium (SWWS-1). 2001*. Stanford, CA USA.

APPENDIX A – SCREENSHOTS

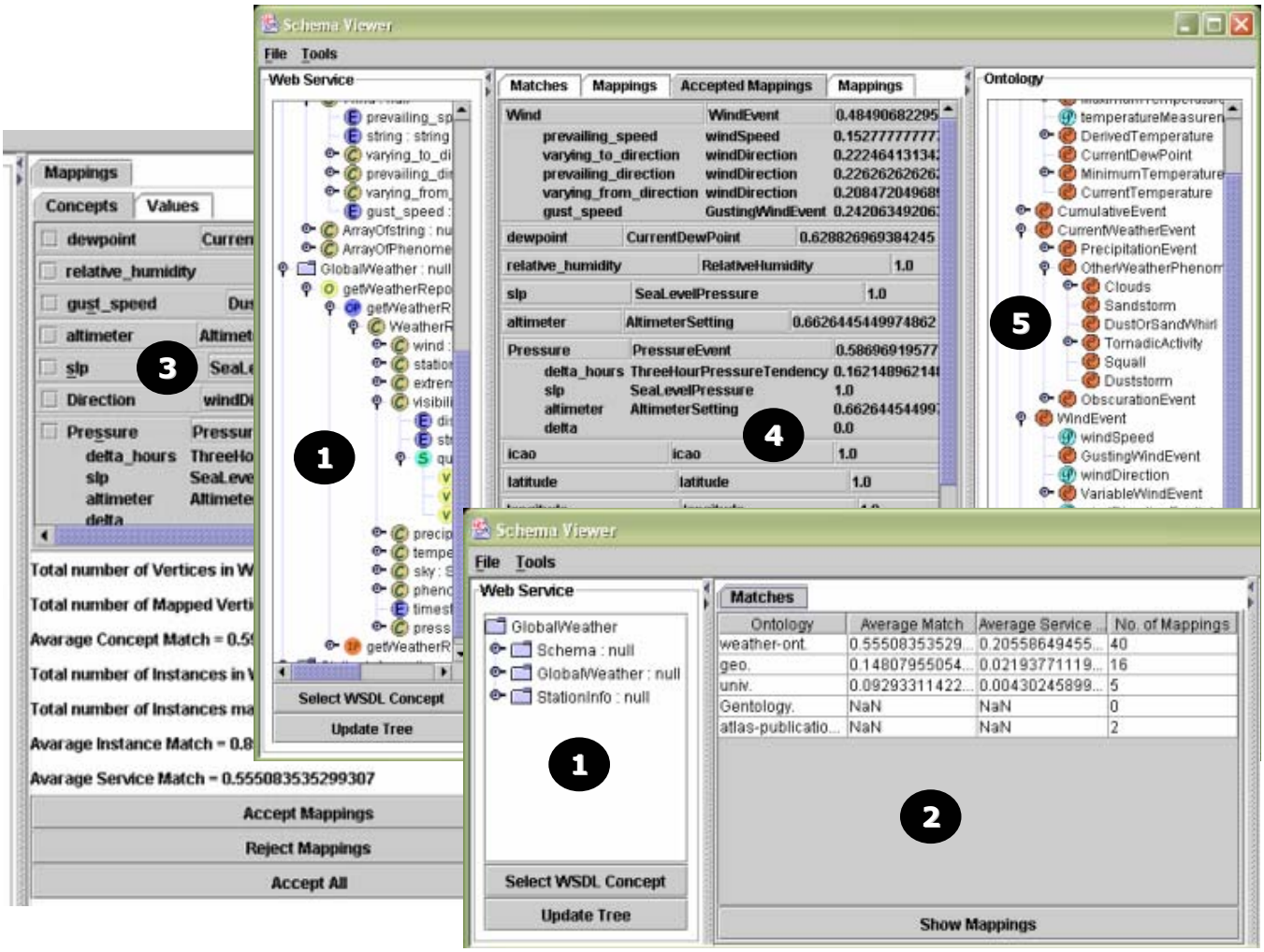


Figure 8. Screenshot of the MWSAF tool

Figure 8 above gives a screenshot of the MWSAF 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 9.

```
+<xsd: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>
```

Figure 9. Part of Annotated WSDL file