Web Service Annotation Using Ontology Mapping

Zhang Duo, Li Juan-Zi, Xu Bin
Department of Computer Science and Technology, Tsinghua University, P.R.China, 100084
zhang-d@mails.tsinghua.edu.cn, ljz@keg.cs.tsinghua.edu.cn, xubin@tsinghua.edu.cn

Abstract

Addressed in this paper is the issue of semantic annotation on Web services. As the popularity of Web services increases, automated discovery and composition of relevant Web services are more and more desired. However, current Web service standards, such as WSDL, are not rich enough to fulfill these tasks, because they cannot specify the semantics in the process of discovery and composition. Thus it is necessary to describe Web services more semantically. Now, OWL is the standard ontology language, and it provides powerful features for expressing such semantics on Web services. In this paper, we describe an approach to annotate Web services with OWL ontology. We formalize the annotation as a process of translating XML schema into an ontology in OWL, mapping this temporary ontology with shared ontologies and generating a semantic description of Web Services with the mapping result.

1. Introduction

Web services, which describe a standard way for integrating Web-based applications, are used as one of the most important means for businesses to communicate with each other (B2B) and to communicate with clients (B2C). Several XML based standards have been proposed to describe Web services. For example, Web Service Description Language (WSDL) has proved very useful in describing the interface of Web services and has been used extensively by industry.

With the growing popularity of Web Services, automated discovery and composition of Web services are highly desirable. However, WSDL is mainly focusing on operational and syntactic details for implementation and execution of Web services. The lack of semantics in WSDL makes it not sufficient to satisfy the requirement of Web service composition and also limits the discovery mechanism of relevant Web services merely on keyword-base search.

Semantic Web, in which information is given explicit meaning, is a vision for the future of the Web making it easier for machines to automatically process and integrate information available on the Web [1]. In Semantic Web, ontologies comprising of concepts with their relationships are the basis for shared conceptualization of a domain. Related researches show that describing Web services with ontologies will enable better discovery and easier composition of Web services [10, 15, 17]. For example, we can use ontology concepts as queries to search Web services. This will be more efficient than just using keywords as queries, because the description and relationship of concepts are explicitly declared in ontologies.

Several approaches are suggested to describe Web services with ontologies [9, 11]. OWL-S [2] is one of the ontology based languages for describing Web services. It proposes to define a set of basic classes and properties for declaring and describing Web services. OWL-S tries to cover the description of Web services in a wide sense, not focus on a particular application domain. However, finding relevant ontologies manually to describe Web services will be a troublesome work, since users have to browse through several domains and a large number of concepts. A classification method is described in [10]. It uses machine learning algorithms to help users simplify the search for relevant ontologies. Another approach aims on adding semantics to current Web service standards [17], such as WSDL and UDDI, to create a better framework for Web service discovery and composition. They propose to annotate WSDL with relevant ontologies and then combine them to a new standard called WSDL-S. To find the relevant ontologies, they use a media structure called SchemaGraph to facilitate the matching between XML schema and ontology [15].

Our approach also aims to enrich WSDL with semantics using ontologies described in OWL. The differences between our approach and [15] are:

Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)
0-7695-2438-9/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

(1) We use a more direct way for the matching between XML schema and ontology. We find that the format of XML schema provides a basis to generate ontology concepts. For example, the relationship between a complexType and its child elements in XML schema is very similar to the relationship between a class and its properties in OWL ontology. This leads us to define several rules on translation from XML schema to OWL ontology. After the translation, the matching between XML schema and ontology becomes the mapping between two ontologies.

(2) We do not introduce a new standard for Semantic Web service. Instead, we generate OWL-S description for Web services from WSDL using our ontology mapping result.

The rest of this paper is organized as follow: We start by reviewing some related work in section 2. In section 3, we outline the architecture of our approach. Section 4 describes the translation from XML schema to OWL ontology. The method of mapping new ontology with existing shared ontologies will be discussed in Section 5. The approach of OWL-S generation will be described in Section 6. Section 6 presents our experiment and final results. We conclude in Section 8.

2. Related Work

Describing Web services using an upper ontology is an active research field currently. Another exciting research field is ontology alignment which aims to solve the heterogeneity problems on the Semantic Web. Our work combines these two fields together to give a method which annotates Web services with semantics. In this section, we explore some related works in these two fields since both of them are very important.

2.1 Semantic Web services

OWL-S [2] is one of the ontology based languages for Semantic Web services. It builds on OWL.

OWL (Web Ontology Language) is a semantic markup language for publishing and sharing ontologies on the Web [1]. It can be used to describe classes and relations between them that are inherent in Web document and applications. Now, OWL is a component of the Semantic Web activity.

OWL-S aims to realize Semantic Web services by providing appropriate semantics enriched descriptions to Web services. It is an ontology of services and its main three parts are: ServiceProfile, ServiceModel, and ServiceGrounding. OWL-S proposes to fulfill a set of work, such as Web service discovery, invocation and composition. Our approach generates an OWL-S description for Web service from the current Web service standard WSDL.

2.2 Ontology Alignment

Ontology alignment has many usages, such as transform one source into another, creating a set of rules between the ontologies, or displaying the correspondences. Our work can also be viewed as an application of ontology alignment. There are several approaches have been studied in this area. Anchor-PROMPT [12, 13] is an ontology merging and alignment tool with a sophisticated prompt mechanism for possible matching terms. It uses a string-based algorithm and refines the results based on users' feedback. Some other studies focus on the structure of the ontology entities. In [16], the constraints of properties, such as range, cardinality and transitivity, are used to facilitate the similarity calculation between two ontologies. More structure information is considered in [3], which uses a method focusing on entities' positions in the path from the root to the entities. Like in many other fields, machine learning methods are also useful in ontology alignment. Some techniques are discussed in [4, 18]. In [5, 6], a machine learning method GLUE is suggested. It first generates a similarity matrix based on the probability which is learned from the distributions of classes in each ontology. Then it produces an alignment from the similarity matrix by using heuristic rules for choosing the more likely correspondences. Compared with these methods discussed above, our mapping method is mainly based on morphological and local structural similarities of two ontologies.

3. Architecture

Our approach can be divided into three steps.

- Translate XML schema into a temporary ontology.
- Map this temporary ontology with existing shared ontologies.
- Preserve the mapping result and use it to generate OWL-S files from the WSDL file.

Figure 1 shows an overview of the architecture. The translation in the first part is based on several rules which make a one-to-one correspondence between XML schema elements and OWL ontology concepts. The structure of the XML schema will be reserved. Elements' names with the targetNamespace of the XML schema will be translated into URIs in OWL

IEEE COMPUTER SOCIETY

ontology. The result of the translation is a temporary OWL ontology.

In the second step, we make use of both morphological and structural similarity for finding the mapping between the temporary ontology and existing shared domain ontology (we use existing ontology for short hereafter). The result of the second step is a mapping file. In the mapping file, every temporary ontology concept will make a pair with its matching concept in existing ontology. The result can also be viewed as the mapping between XML schema and existing ontology, since the translation in the first step is a 1:1 correspondence.

In the third step, an OWL-S description of the Web service will be generated from the WSDL file using the mapping result. The description contains three parts: ServiceProfile, ServiceModel, and ServiceGrounding. We also preserve the mapping file as one part of the final result.

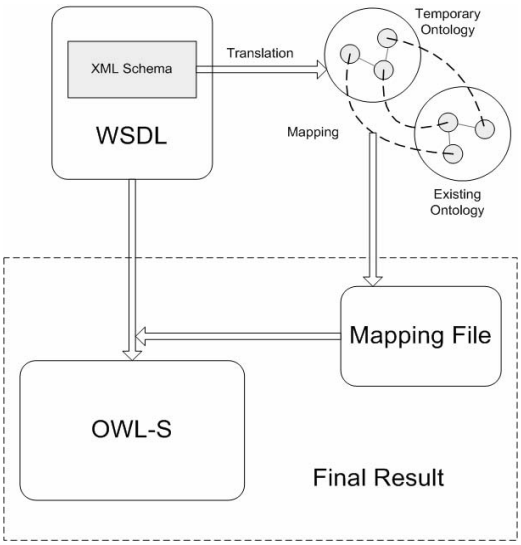

Figure 1.Overview of architecture

4. XML schema to Ontology

XML schema and OWL ontology share a common foundation: a hierarchy structure. The translation in our first step is mainly based on this foundation. To make the description more clearly, we first define some notations. In XML schema, the elements which appear as the direct children of schema are called *global elements*. This is the same as the definition in [7]. For the other elements, we called them *local elements*. *Sub-elements* of a complexType indicate the direct child elements of that complexType. And a complexType is called the *super-structure* of its sub-elements. Our translation rules are listed as follows:

Rule 1. the targetNamespace of XML Schema is translated into the namespace of OWL

This rule ensures that all the names of elements in XML schema can be used directly in the OWL ontology. Note that if there is some naming conflict, we can add different numbers after each name to distinguish them.

Rule 2. complexType is translated into owl:Class

This is the most general rule and should be used whenever there is a complexType in the schema.

Rule 3. simpleType is translated into owl:Class

SimpleType in XML schema has various forms, such as list types, union types, a form of enumeration and etc. All of these forms indicate that simpleType has a complex structure and should be translated into an owl:Class. In Fig. 2, we give an example of how to translate a simpleType in form of enumeration into an owl:Class in form of owl:oneOf. The original form of "ForecastDays" is shown in Fig. 4. Each enumeration value of the simpleType is translated into an individual of owl:Thing.

```
<owl:Class rdf:ID="ForecastDays">
 <owl:equivalentClass>
  <owl:Class>
   <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:ID="Day1"/>
    <owl:Thing rdf:ID="Day2"/>
   </owl:oneOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

Figure 2.Enumerated class from simpleType

Rule 4. global element is also translated into owl:Class

Since a global element always has an anonymous type of complexType or simpleType, it should also be translated into an owl:Class which ID is the element's name.

```
<owl:ObjectProperty rdf:ID="DayToBroadcast">
 <rdfs:range rdf:resource="DayInfo"/>
 <rdfs:domain>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="GetWeatherByZip"/>
    <owl:Class rdf:about="GetWeatherByCity"/>
   </owl:unionOf>
  </owl:Class>
 </rdfs:domain>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="City">
 <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 <rdfs:domain rdf:resource="GetWeatherByCity"/>
</owl:DatatypeProperty>
```

Figure 3.Properties in the translation

IEEE COMPUTER SOCIETY

Rule 5. local element with a type of simple type which is built in to XML Schema is translated into owl:DatatypeProperty

In Fig. 4, the local element "City" is translated into a DatatypeProperty shown in Fig. 3. The type of the local element becomes the range of the DatatypeProperty, and the super-structure of the local element becomes the domain of that DatatypeProperty.

Rule 6. local element with a type of complex type definition in XML Schema, such as complexType and simpleType, is translated into owl:ObjectProperty

The element "DayInfo" in Fig. 4 is one example under this rule. One characteristic of the element is that it has two super-structures. When it is translated into an owl:ObjectProperty, its domain should be an owl:unionOf two classes as shown in Fig. 3.

```
<s:schema elementFormDefault="qualified"
          targetNamespace="http://keg.cs.tsinghua.edu.cn/">
 <s:element name="GetWeatherByZip">
  <s:complexType>
   <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
               name="PostalCode" type="s:string" />
    <s:element name="DayToBroadcast" type="tns:DayInfo" />
   </s:sequence>
  </s:complexType>
 </s:element>
 <s:element name="GetWeatherByCity">
  <s:complexType>
   <s:sequence>
    <s:element name="City" type="s:string" />
    <s:element name="DayToBroadcast" type="tns:DayInfo" />
   </s:sequence>
  </s:complexType>
 </s:element>
 <s:complexType name="DayInfo">
  <s:sequence>
   <s:element name="Day" type="s:string" />
   <s:element name="Date" type="s:string" />
  </s:sequence>
 </s:complexType>
 <s:simpleType name="ForecastDays">
  <s:restriction base="s:string">
   <s:enumeration value="Day1" />
   <s:enumeration value="Day2" />
  </s:restriction>
 </s:simpleType>
</s:schema>
```

Figure 4.Example in XML Schema

Rule 7. the translation of attribute is the same as local elements

This rule indicates that both the attribute and local element can be viewed as properties of class.

Rule 8. local element with an anonymous type is translated into owl:Class

The element's name will be the ID of that owl:Class. An anonymous type always has a complex structure and contains some semantic information. So this translation is necessary.

Rule 9. minOccurs and maxOccurs are translated into owl:minCardinality and owl:maxCardinality respectively, and add these restriction to the relevant owl:Class

Our translation is mainly based on these general rules. According to these rules, the translation makes a one-to-one correspondence between the XML schema and OWL ontology on both the morphological and structural aspects. We will show that such a correspondence plays an important role in next two steps.

5. Mapping Ontologies

Ontology mapping, which is also known as ontology alignment, is an active topic in Semantic Web. The ontology alignment problem can be described as: given two ontologies each of which describes a set of discrete entities (classes, properties, etc.), find the relationships holding between these entities.

For every two entities, our ontology mapping method concerns two aspects of them: terminology similarity and structural similarity. Terminology similarity is to compare the morphological similarity between the names of two entities by a measure function. In our approach, we use Levenstein distance as the measure function. The structural similarity of two entities will be calculated based on the terminology similarity of their relative entities. Then we give a similarity score for the two entities which is calculated by the weighted average of their terminology similarity and structural similarity.

5.1 Terminology Similarity

We use edit distance to calculate the terminology similarity of entities. Specially, we use Levenstein distance here.

Generally speaking, an edit distance of two objects is the minimal cost of operations applied to one of the objects for obtaining the other. For edit distance on strings that transform s to t, the operation set Op will be:

- Copy a character from s over to t
- Delete a character in s
- Insert a character in t
- Substitute one character for another

A formal definition of edit distance is:

IEEE COMPUTER SOCIETY

Definition. Given a set Op of string operations, and a cost function $\omega : Op \to R$, such that for any pair of strings (s,t), there exist a sequence of operations which transforms the first one into the second one, the edit distance $\delta(s,t)$, is the cost of the less costly sequence of operations which transform s in t

$$\delta(s,t) = \min_{(op_i)_I ; op_n(...op_1(s))=t} (\sum_{i \in I} \omega_{op_i})$$

Levenstein distance is the edit distance with all costs to 1. From the definition, we can see that the lower the value of $\delta(s,t)$ is, the higher similarity will s and t have. Therefore, we define the function $TermSim(s,t)$ to calculate the similarity of two strings:

$$TermSim(s,t) = \frac{MaxLength(s,t) - \delta(s,t)}{MaxLength(s,t)}$$

Here, $MaxLength(s,t)$ represents the max length of strings *s* and *t*.

Table 1. Terminology similarity of ontologies

For each $class_i \in TO$

  For each $class_j \in EO$

    Output $TermSim(name(class_i), name(class_j))$;

For each DatatypeProperty $DP_i \in TO$

  For each DatatypeProperty $DP_j \in EO$

    Output $TermSim(name(DP_i), name(DP_j))$;

For each ObjectProperty $OP_i \in TO$

  For each ObjectProperty $OP_j \in EO$

    Output $TermSim(name(OP_i), name(OP_j))$;

The algorithm for calculating terminology similarity of temporary ontology (TO) and existing ontology (EO) is described in Table 1. In the table, we use $Name(classT)$ to denote the name of $class\ T$.

The algorithm calculates the terminology similarity of every two entities of belong to same category. For example, each class in temporary ontology will be compared with all the classes in the existing ontology but not compared with the properties in the existing ontology.

5.2 Structural Similarity

Ontology is not a set of entities separately but an integrity of entities with their relationship. So we consider the structure information is another kind of important information for ontology mapping. For example, if the properties of one class have good matches with the properties of another class, there is a higher probability that these two classes are to be mapped.

In our algorithm, we consider the *neighbors* of each entity. A *neighbor* of an entity *E* is the entity which has a direct relationship with *E*. For an owl:Class, its properties, super classes, sub classes are called its neighbors. For a property, its domains and ranges are called its neighbors. The algorithm for calculating structural similarity of class entities is shown in Table 2. The algorithm for properties is similar.

Table 2. Algorithm for structural similarity

For each $class_i \in TO$

  For each $class_j \in EO$

    If $(p < q)$

      For every $\{i_1, i_2, ..., i_p\} \subset \{1, 2, ..., q\}$

        $StructSim(class_i, class_j)$

        $= \max(\sum_{k=1}^{p} TermSim(NT_k, NE_{i_k})) / p;$

    Else

      For every $\{i_1, i_2, ..., i_q\} \subset \{1, 2, ..., p\}$

        $StructSim(class_i, class_j)$

        $= \max(\sum_{k=1}^{q} TermSim(NT_{i_k}, NE_k)) / q;$

    Output $StructSim(class_i, class_j)$;

We use $StructSim(u,v)$ to identify the structural similarity of two entities *u* and *v*. We also use TO to identify temporary ontology and EO for existing ontology here. The neighbors of $class_i \in TO$ are the set $\{NT_1, NT_2, ..., NT_p\}$ and the neighbors of $class_j \in EO$ are the set $\{NE_1, NE_2, ..., NE_q\}$.

The algorithm gives the best terminology similarity among the neighbors of two entities as their structure similarity. We can also see that the algorithm just focus on a local structure of the ontology, since it only concerns the neighbors of each entity. An example of structure similarity calculation is given in Table 3. The ObjectProperty *DayToBroadcast* in the example is described in the Fig. 3 and the ObjectProperty *DayForcast* is defined in an existing ontology. For all the matches between neighbors of *DayToBroadcast* and *DayForcast*, the best match which has the

IEEE COMPUTER SOCIETY

maximum sum of terminology similarities is {(DayInfo, Date), (GetWeatherByZip, WorldWeather), (GetWeatherByCity, DailyWeatherReport)} as shown in Table 3.

Table 3. Example of structure similarity

Temporary Entity	Neighbors	Best match	neighbors	Existing Entity
DayTo Broadcast	DayInfo	0.286	Date	Day Forcast
	GetWeather ByZip	0.333	World Weather	
	GetWeather ByCity	0.389	Daily Weaterh Report	
StructSim: (0.286 + 0.333 + 0.389) / 3 = 0.336				

5.3 Similarity score

The similarity score of two entities u and v is calculated as the weighted average of their terminology similarity and structural similarity. The formula is defined as:

$$SimScore(u,v) = \lambda_1 * TermSim(u,v) + \lambda_2 * StructSim(u,v)$$

where $\lambda_1 + \lambda_2 = 1$. In our approach, we give λ_1 a value of 0.6 and give λ_2 a value of 0.4.

For each entity E in a temporary ontology, the similarity scores with all entities in the existing ontology will be calculated and ranked. The entity in the existing ontology which has the highest similarity score with E is selected as the matching entity for E.

Fig 5. shows an example of mapping results. The snippet gives a pair of mapping entities and their similarity score.

```
<map>
  <Cell>
    <entity1 rdf:resource='Location'/>
    <entity2 rdf:resource='LocationCode'/>
    <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>
      0.8574589074186623
    </measure>
  </Cell>
</map>
```

Figure 5. Example of mapping result

6. OWL-S generation

OWL-S uses three types of knowledge to describe a Web service: *ServiceProfile*, *ServiceModel*, and *ServiceGrounding*. *ServiceProfile* is used to describe what a Web service does, *ServiceModel* describes how it works and *ServiceGrounding* is used to specify how to access it.

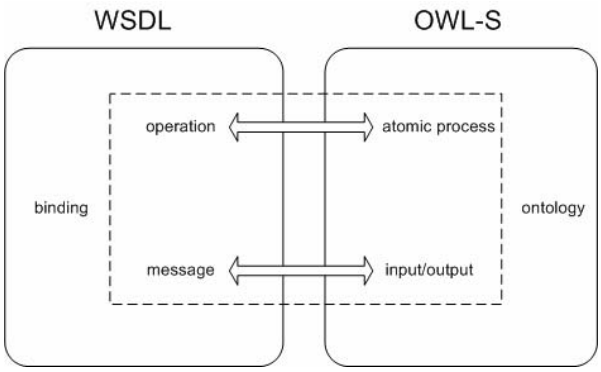

Figure 6. Relationship between WSDL and OWL-S

Fig. 6 shows the relationship between WSDL and OWL-S. WSDL and OWL-S share a number of common points in describing a Web service process. WSDL uses operations and messages to describe a Web service process, while OWL-S ServiceModel uses atomic processes and inputs/outputs to describe the process. Though their names are different, their effects are the same.

There are two main differences between WSDL and OWL-S:

1. *WSDL uses XML Schema to characterize the input and output messages of Web services, whereas OWL-S uses OWL classes.*

2. *OWL-S has no means to express the binding information that WSDL provides.*

The first difference is the reason why WSDL lacks of semantics. Our translation and ontology mapping method fill up this gap, since we can find OWL classes which match with the XML schema types to describe the input and output messages instead of XML schema types themselves For the second difference, [2] suggests to use WSDL as the grounding mechanism for OWL-S. The basic idea is: First, it uses an *owl-s-parameter* attribute to describe a *part* of message in WSDL instead of *type*. Then, in *ServiceGrounding* it uses a *WSDLGROUNDING* class, which is a subclass of *Grounding,* to refer the relevant construct in WSDL.

Based on all of these similarities and ideas, we define certain rules for the generation of OWL-S description of Web services:

- A portType in WSDL will become a ProcessModel in ServiceModel.
- An operation in WSDL will become an atomic process in ServiceModel.
- The input and output messages of an operation will become the inputs and outputs of an atomic process respectively
- The mapping result of a message part's type will become the owl-s-parameter for that message part.

IEEE COMPUTER SOCIETY

According to these rules, the third step of our approach completes the generation of ServiceModel and ServiceGrounding of OWL-S. It also covers a part of ServiceProfile, that is the input and output information of the Web service. The rest part of ServiceProfile, such as *contactInformation* and *qualityRating*, will be added manually.

Our final result also preserves the mapping result, since it includes all the mapping information not only for classes but also for properties. We consider it very useful for future work, such as the service matching mechanism in service discovery and service composition.

7. Experiment

In the first two steps, our approach completes a process of mapping XML schema with ontology concept. The first step makes a one-to-one correspondence between XML schema and temporary ontology. The second step finds the mapping between the temporary ontology and existing ontologies. In this section, we will show the ability of our approach by presenting some of our experiment results.

7.1 Dataset

For the number of our domain specific ontologies is limited, we just focus on two domains: travel domain and weather domain. Our corpus is made up of 49 Web services of travel domain and 16 Web services of weather domain. All these files are selected from a group of totally 856 Web services from Amazon.com and XMethods.com. We also create a travel domain ontology and a weather domain ontology according to prior knowledge. To make our ontologies more general, we select 20 files randomly from all the 65 Web service files to modify our ontologies (5 from weather domain and 15 from travel domain). We call these files *modification files*. The other 45 files are called *independent files*. Before the experiment, we first annotate all the service files manually. For each element in the XML schema, we find the most appropriate concept in the domain ontology as the mapping with that element.

7.2 Evaluation Measures

We evaluate our results by conducting the measurement of *precision*. It is defined as follow:

$$Precision = \frac{correct\ discovery\ mapping}{total\ number\ of\ mapping}$$

Since each element in the XML schema has exactly one correct mapping, we only use *precision* here.

7.3 Experiment Result

The experiment results are shown in Table 4.

Table 4. Mapping result in terms of precision(%)

	Travel Domain	Weather Domain
Modification files	92.5	96.4
Independent files	78.6	79.5
Average	83.6	86.5

The average result is the experiment on both the modification files and independent files. From the result we can see that:

(1) The modification files have more precision than the independent files. To make our ontologies more general, we use the modification files to modify the ontologies making sure that for each XML schema element in modification files there really exits an ontology concept matched with that element. Otherwise, we add or change some structures of the ontologies. So the modification files have more similarity with the ontologies after modified. For the independent files, their XML schema structures and elements' names are much more different from the ontologies, so the precision is not high.

(2) The result of weather domain is better than the travel domain. That is because the weather domain ontology is smaller than the travel domain, and the weather Web services are very similar with each other.

(3) The differences between the results of modification files and independent files also show that if the ontology is created more general, the precision will be improved.

7.4 Complexity Analysis

Here we give the computational analysis of our ontology mapping method.

Suppose the numbers of entities in the temporary ontology and the existing ontology are m and n respectively.

In section 5.1, the computational complexity of Levenstein distance is $O((string's\ length)^2)$. So the total computational complexity of the first part is $O(mn(string's\ length)^2)$. In section 5.2, the structural similarity calculation of two entities can be viewed as a bipartite matching problem. The computational

IEEE COMPUTER SOCIETY

complexity of the second part is $O(mn(neighbor's\ number)^3)$. In section 5.3, since the computational complexity of the ranking algorithm for every entity of temporary ontology is $O(n \log n)$, the computational complexity of this part is $O(mn \log n)$.

The total computational complexity of our ontology mapping method is the sum of the three parts above. One advantage of our approach is that our mapping method can be improved easily as the study of ontology alignment further developed. We are currently working on some other mapping methods which combine advantages of other methods and ours.

8. Conclusion and Future work

In this paper, we discuss a process of annotating Web services with semantics. First, we define several rules to translate a XML schema to ontology in OWL. Then we map the ontology with existing ontologies. Finally we generate an OWL-S description for the Web service using the mapping result. We also carry out experiments to test the mapping method and compare the results. We plan to further improve both the accuracy and efficiency of the annotation as our future work. One attribution of our work is that we give a method which maps an XML schema with an ontology. We find the mapping result contains large amounts of information. This leads us to a further study on how to use such information on Web service discovery and composition.

9. References

[1] OWL: http://www.w3.org/TR/2004/REC-owl-features-20040210/

[2] OWL-S: Semantic markup for Web Services, version 1.1, available at http://www.daml.org/services/owl-s/1.1/

[3] T. L. Bach, R. Dieng-Kuntz, and F. Gandon, "On ontology matching problems (for building a corporate semantic web in a multi-communities organization)", *In Proc. of ICEIS 2004,* Port, Portugal, 2004.

[4] J. Berlin and A. Motro, "Database schema matching using machine learning with feature selection", *14th International conference on advanced information system engineering (CAiSE)*, Toronto, Ontario, Canada, 2002, pp.452-466

[5] A. Doan, P. Domingos, and A. Halevy, "Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach", *In Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001

[6] A. Doan, J. Madhavan, P. Domingos and A. Halevy, "Learning to Map Between Ontologies on the Semantic Web", *WWW2002*, Honolulu, Hawaii, USA, May 2002

[7] D. C. Fallside and P. Walmsley, XML Schema Part 0: Primer Second Edition, available at http://www.w3.org/TR/xmlschema-0/

[8] M. Ferdinand, C. Zirpins, and D. Trastour, "Lifting XML Schema to OWL", *Web Engineering - 4th International Conference (ICWE 2004)*, Munich, Germany, July 2004, pp. 354-358

[9] R. Herzog, H. Lausen, D. Roman, P. Zugmann (eds.): WSMO Registry. WSMO working draft, available at http://www.wsmo.org/2004/d10/v0.1/, 2004

[10] A. Hess and N. Kushmerick, "Learning to Attach Semantic Metadata to Web Services", *Proceeding of the 2nd International Semantic Web Conference (ISWC 2003)*, Florida, USA, Oct. 2003, pp. 258-273

[11] R. Lara, D. Roman, A. Polleres and D. Fensel, "A Conceptual Comparison of WSMO and OWL-S", available at http://www.wsmo.org/2004/d4/d4.1/v0.1/20050106/

[12] N. F. Noy and M. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment", Proceedings of the National Conference on Artificial Intelligence (AAAI 2000), Austin, Texas, USA, 2000, pp. 450-455

[13] N. F. Noy and M. Musen, "Anchor-PROMPT: Using Non-local Context for Semantic Matching", *In Proceedings of IJCAI 2001 Workshop on Ontology and Information Sharing*, Seattle, Washington, USA, 2001, pp.63-70.

[14] M. Paolucci, N. Sriivasan, K. Sycara and T. Nishimura, "Towards a Semantic Choreography of Web Services: from WSDL to DAML-S", *Proceedings of the International Conference on Web Services (ICWS 2003)*, Las Vegas, Nevada, USA, June 2003, pp. 22-26

[15] A. Patil, S. Oundhakar, A. Sheth and K. Verma, "METEOR-S Web Service Annotation Framework", *WWW2004*, New York, USA, pp. 553-562.

[16] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching", *The VLDB Journal*, Springer-Verlag, New York, USA, Dec. 2001, pp.334-350.

[17] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller, "Adding semantics to web services standards", *In Proceedings of the 1st International Conference on Web Services (ICWS'03)*, Las Vegas, Nevada, USA, 2003

[18] G. Stumme and A. Mädche, "FCA-merge: bottom-up merging of ontologies", *In Proc. 17th IJCAI,* Seattle (WA US), 2001, pp. 225–230

IEEE COMPUTER SOCIETY