Fourth International Conference on Semantics, Knowledge and Grid

Modeling Semantic Web Service using Semantic Templates

Liang Hu[#1], Jian Cao[#2], Zhiping Gu[*3]

[#] *Department of Computer Science and Technology, Shanghai Jiaotong University*
200030 Shanghai, China
[1]milkrain@sjtu.edu.cn, [2]cao-jian@sjtu.edu.cn

[*] *Department of Electrical Engineering, Shanghai Technical Institute of Electronics & Information*
201411 Shanghai, China
[3]01019@sheic.edu.cn

Abstract: Web Services have been widely used recent years. In order to enable the intelligent discovery and use by machine, the semantic information should be provided to represent various aspects of Web Services. If it always need build complex semantic information from scratch to describe every aspect of Web Service, it will lead to not only large redundancy and inconsistency but also low maintainability and extensibility. The Customizable Semantic Template is proposed to resolve such issues, which enables to semantically model any aspect of Web Service in a more flexible and efficient way. A universal method to automatically generate Semantic Template instance is also proposed to resolve the issues like high workload for building semantic information for every aspect of Web Service manually and high specialized domain knowledge required.

I. Introduction

Web Service has been widely used in a lot of domains, due to the powerful interoperability for operating, accessing and sharing resource. The number of Web Services is growing rapidly in every domain. As a result, it becomes obviously unfeasible to organize, classify and manage manually. From the automation perspective, the most concerned is how to enable machines to aware, manage and operate Web Services automatically. In order to achieve such goal, the semantic technology for Web Services has begun to research.

Semantic information is the foundation to enable machines to be intelligent. There are already some researches to generate semantic information for Web Services in automatic or semi-automatic ways. Think about the following questions, Q1: whether the more semantic information created for any aspect of Web Services the better it will be? Q2: How to appropriately create semantic information for different aspects of the Web Services? Q3: Whether it is necessary to create independent and complex semantics for each different aspect of Web Services from scratch?

Our view paper for above questions is: For Q1, the answer is negative, the more semantic information created the more time will consume, and the number of errors will increase. However, to an application, it often only focuses on certain kinds of semantic information and ignores the others. Furthermore, sometimes it is necessary to balance the precision and recall, too much semantic information may make agent confused for reasoning. For Q2, it is a further supplement for Q1. Creating semantic information for Web Service should follow the principle that it should be high cohesion instead of providing some kind of semantic information could cover data, operation, and error handling all together in one place. For Q3, the answer is negative either. It can further clarify the meaning of the Q2. There is more or less duplicated semantic information created for various aspects of the Web Services, if every aspect always creates all of the semantic information from scratch. Finally, it will probably result in much redundancy and inconsistency and widen the gap of collaboration and integration. To avoid these issues, it should be possible to reuse existing semantic information to describe various aspects of the services. The semantic description for certain aspect of Web Services may be too complex to describe with simple semantic information, but it is able to construct composite semantic information for describing by composing the existing simple semantic information from multiple basic aspects.

In this paper we proposed a formal solution to these issues — Semantic Template. For different aspects of Web Services, we can customize different types of Semantic Templates as the semantic representation. Our research work mainly covers two areas: 1) Customize various types of Semantic Templates for modeling all aspects of Web Services. 2) Construct a flexible framework to automatically generate Semantic Template instances.

In Section 2, the usage and features of different type Semantic Template is presented. Section 3 presents the key technologies in the process of automatically generating Semantic Template instance. Section 4 presents the experiment and evaluation base on the prototype. Section 5 lists the related works. Section 6 presents conclusions and the outline for future work.

II. Modeling Semantic Templates

As previously discussed, if it always needs to build complex semantic description for each aspect of Web Service from scratch, it will generate a lot of redundant and inconsistent semantic information and lead to low maintainability and extensibility. Therefore, it is necessary to find a reusable mechanism to improve the flexibility and availability in use of the semantic information.

Semantic Templates are created for modeling any aspect of the Web Service semantically. Semantic Template schema is defined to model the profile of some aspect of Web Services.

978-0-7695-3401-5/08 $25.00 © 2008 IEEE
DOI 10.1109/SKG.2008.52

The Semantic Template instance is created to describe this aspect of some concreted Web Service with the concrete semantic information according with the content as schema defined.

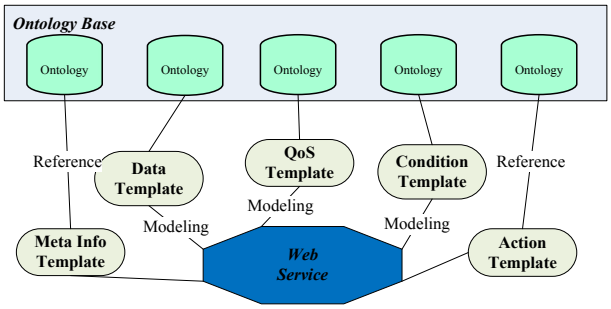

Fig. 1 Semantic Templates for modeling Web Services

As shown in Fig.1, Semantic Templates can be used to modeling any aspect of Web Services, which provide a reasoning context to make agents be able to aware what kind of behaviour should cause. There are some important features for Semantic Template:

High cohesion: Semantic Template should be the high cohesive. Each basic Semantic Template should focus on describing a point of function or attribute of Web Service. A complex Semantic Template should refer a couple of existing Semantic Templates to express sophisticate semantics.

Composite ability: Semantic Templates are compostable. Complex semantic information for describing some aspect of Web Service usually could be decomposed into more basic semantics, so a couple of basic Semantic Templates can be used to compose a composite Semantic Template. Furthermore the composite Semantic Template can be used to construct more complex templates.

Extensibility: Semantic Template is extensible. The way to extend Semantic Template through composing templates has been presented. Another way is to extend Semantic Template through inheritance, which can add additional types of semantics for the base template.

Virtuality: Semantic Template makes it possible to use Web Service virtually. The Web Services that exposed to client probably may not physically exist in system. Semantic Templates may have decorated the inputs and outputs or encapsulated the internal business process. The difference between physical Web Service and virtual Web Service are transparent to client, because agents can handle the internal process with the Semantic Template support.

Dynamic: Semantic Template can represent dynamic values or logics. It could express not only the logic for validating input and output at run time but also the abstract business processes logics for discovering or executing on the fly.

1.1. Modeling Basic Semantic Templates

Since some basic semantics can be composed to describe some complex aspect of Web service, some types of basic Semantic Templates should be constructed first to model the most basic aspects of the Web Services. Such templates can be called Meta Semantic Templates which must be high cohesive because they are the basic building blocks for constructing complex Semantic Templates.

Semantic Template does not specify the language to construct it. Such as RDF/S, OWL can be used to construct it as shown in Fig. 2, and it can also used WSMO[10], SWSO[17] to construct the logics.

A. Action Template: Generally, the purpose of an operation in Web Service can be described as an action. The most common element to form an action is a verb and its target. For example, "Reserve Hotel", {Verb = "Reserve", "Target = Hotel"}. So we can define the Action Template with the semantic properties of {Verb, Target} to describe the basic behaviour of the operation semantically. Fig. 2 shows a very simple Action Template and its instance. Although the Action Template we defined only contains two properties, it is competent for satisfying the request of the Web Service discovery for users.

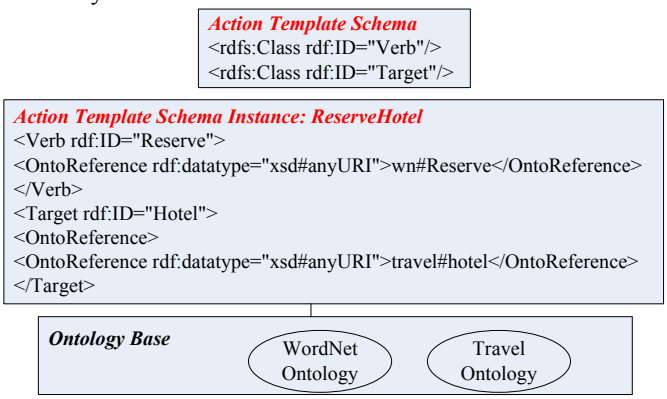

Fig. 2 Action Template Schema and Instance[1]

B. Meta Information Template: Meta information for Web Service should always be provided for better use and management such as category, providers, version, which could lead users or agents to identifying Web Services more directly. So the Meta Information Template can be created to accommodate such information. Since Semantic Template is extensible, some organization can extend Meta Information Template schema with the specific information. For example, add public key as the additional attribute to the template for some high security demanded environment.

C. Data Template: The basic function of Web Services is to access data. Several Web Services can be composed by data stream, so Data Template can be defined for data name and data type. Data Template can contain sub Data Templates as properties, because the complex data concept may contain some simple data concepts. Data Template not only can express the concept of data, but can be referred by other Semantic Template. For instance, it can be composed in Condition Template Data Template for precondition or post-condition checking.

D. QoS Template: In the complex network environment, the QoS for Web Services is always concerned. QoS Template can be used to inform agent to create Web Services execution context, such as the security context, cost, response time. QoS Template can be overridden, for example, global environment

[1] The following figures will no longer involve language details for Semantic Templates modeling, instead they only illustrate the outline of the structures.

may define the security context using Triple DES encryption while certain operation should use RSA encryption, and the encryption of global QoS Template will be overridden. QoS Template is usually composed into some complex Semantic Template for further use.

1.2. Modeling Advanced Semantic Templates

According to the principle of high cohesion, it is a wrong way to construct a Semantic Template with all kinds of semantics and logics. The correct way is to take advantage of the composite ability to create a complex template by composing various existing templates. Another benefit for such method is that agent does not need to handle all complex reasoning tasks for the complex template, and it could delegate the reasoning task to each sub Semantic Template processor.

A. Condition Template: Sometimes it needs to check validation or do some environment setting before running Web Services, and to verify the result or release some resource when it finishes running. Condition Template could be defined to describe such requirement. For instance, precondition property defines the expected inputs and post condition defines the expected result. QoS Template can be composed into Condition Template to set the execution environment, such as security context, transaction. Data Template can also be referred for data validation, result verification, and data transformation.

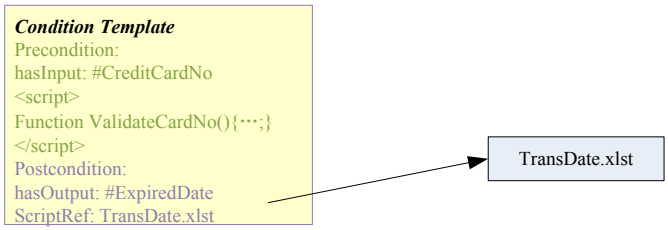

Fig. 3 Condition Template Sample

Condition Template describes some properties which often needs to validate data, check constrain at runtime. Usually, Condition Template instance should provide the rules or logic by itself. As shown in Figure 3, the logics are built in by such as XLST with XQuery, JavaScript, SWSL[16] to enable agent run this logics for checking in the specified scenarios.

B. Operation Template: Each operation can be looked as a minimum Web Service. The most basic elements to describe an operation are: operation name, inputs, and outputs. The name of operation literally implies the purpose of the operation, so obviously Action Template can be used for modeling which. Data Templates used to describe the input / output data items. Operation must be able to provide the meta info to identify itself if agent or client requests, so Meta Information Template can be referred to descript it. Condition Template can also be composed into Operation Template if it needs to do some work such as input/output validation, data transformation and execution context setting.

C. Regular Process Template: Consider following situations: 1) To finish a task always need follow some regular process. 2) There is a complex process to run, but it is no need to make client involve the detail and only a simple access interface is required. 3) It need attach some additional services such as transaction or encryption to the business service at runtime, but it mustn't couple with business logic.

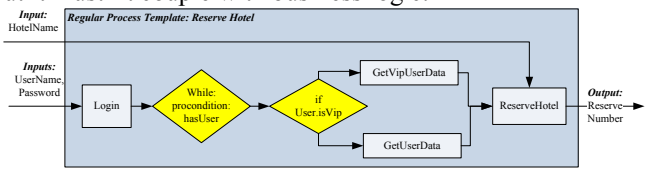

Fig. 4 Regular Process Template: ReseverHotel

For such above requirements, it need construct a process to driven the Web Services flow and provide a simple interface for access. Such process can be model by Regular Process Template, as shown in Fig. 4, which is composed of a series of Operation Template instances with control semantics.

The Regular Process Template instance provides agent a state machine, which enable agent to drive the whole process with the state transfer by checking the inputs/outputs or precondition/postcondition of the Web Services at runtime.

D. Abstract Business Process Template: Many standard or commonly used business processes can be found in each domain, which facilitate better collaboration inside domain and easier accessing outside domain. A typical example is "withdrawal from ATM": any ATM always can provide the same process for withdrawing regardless whichever bank your credit card belongs to.

Base on above idea, we can model such kind of process using Abstract Business Process Template. An abstract process is composed of a series of activities, so Action Template and Condition Template can be composed as Activity Template to model the activities, where Action Template defines the expected behaviour and Condition Template defines the precondition and post condition for every activity in the abstract process. Therefore, a sequence of Activity Template instances can be composed to define an Abstract Business Process Template instance.

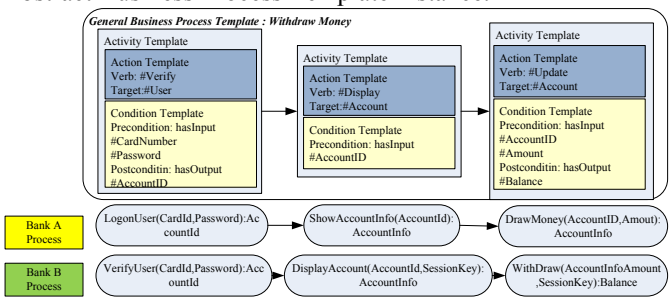

Fig. 5 Abstract Business Process Template Instance: Withdraw Money

As shown in Fig. 5, unlike Regular Process Template modeling a regular flow for concrete Web Services, Abstract Business Process Template is commonly used to model abstract processes without involving implementation, so it makes possible to discover and compose Web Services on the fly across any organization which supports such business process in some domain. Comparing with Regular Process Template, Abstract Business Process Template enables a more flexible and abstract way to run the Web Services on the fly, because agents will pick up services at runtime. However, the

dynamic discovery, composition and invocation are possible lead to lower performance and increasing the risk of failure.

E. Virtual Web Service Template: Sometimes the physical Web Services in the system cannot work well in some situations. Consider the following situations: 1) Some of the processes are unstable, so the flow of Web Services works for this process is also changeful. It need provide a stable access point to ignore the internal unstable processes. 2) For the Process Template such as Regular Process Template, Abstract Business Process Template, it need provide a service interface to enable work as normal Web Service. So all kinds of semantics such as operation, precondition also need construct.

To address these situations, Virtual Web Service Template can be created for modeling the virtual Web Service. Virtual Web Service is used to encapsulate the internal complexity and uncertainty, and it always provides a simple interface for access just like using a physically existing Web Service.

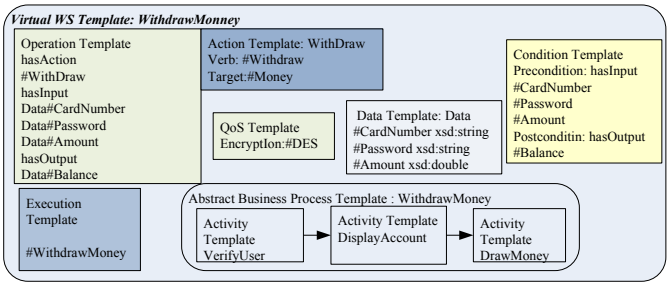

Fig. 6 Virtual Web Service Template Instance: WithdrawMoney

Fig. 6 shows a Virtual Web Service Template instance which encapsulates the business process "WithdrawMoney". A Virtual Web Service Template instance can accommodate all types of Semantic Template instances for describing a virtual Web Service. With these Semantic Templates, agents can treat a virtual Web Service the same as a normal Web Service for reasoning and it is able to construct and run the virtual Web Service on the fly according to the execution semantics.

Virtual Web Service Template and Process Template supplement each other. Sometimes agents need run a Virtual Web Service to act as an activity defined in a Process Template instance. Vice versa, Process Template can be used to construct execution semantics for Virtual Web Service Template. Regular Process Template can model the clear process for Virtual Web Service, while the abstract or general process can be modelled by Abstract Business Process Template. Even the process changed it hardly affect client, because client only know the interface of the Virtual Web Service instead of coupling with the internal process.

III. Automate Semantic Template Instances Generating

Since all types of Semantic Templates have been constructed for modeling Web Service, but it is unfeasible to build out Semantic Template instances absolute manually for the mass of Web Services. This section will discuss some key technologies for analysing and processing corpora so as to generate the Semantic Template instances automatically.

3.1. Corpora for Generating Instances

In the process of building Web Services a lot software products were created, such as WSDL, UML production and requirement documents. Since each kind of software product has different usage, it can be used to generate different types of Semantic Template. Fig. 7 shows some software products corpora used to generating different types of Semantic Template instances.

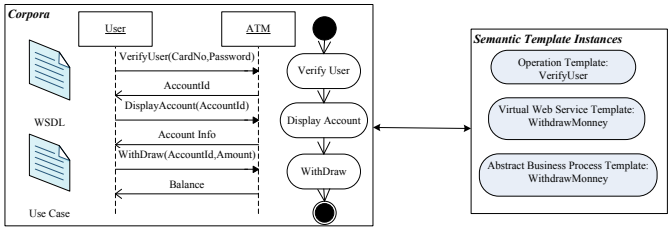

Fig. 7 Corpora for Generating Semantic Template Instances

Due to the time and resource limitation, our research for automatic generating Semantic Template instance is major based on analyzing the WSDL files. Although the WSDL files are our major research corpora, the methods following discussed are universal methods to generate Semantic Template instances from multiple corpora.

3.2. Text Preprocessing

Some of the software products described in natural language such as requirement or use case documents can skip this step and direct input for natural language processing, but some others are not accord with the natural language form. For instance, the operation name and data type name in WSDL do not have any separator, and the machine cannot handle such text. However, if it does some preprocessing over such text, it could be transformed into natural language text.

A. Text Preprocessing Library: Although the kinds of software products are different, a series of text normalization processing steps can always be built to output the text of natural language form. Therefore it is necessary to create a library to save preprocessing algorithms. Table 1 lists some commonly used algorithms for preprocessing.

Table 1 Preprocessing Library

Algorithm	Configuration	Comment
Naming Convention Segment	Segment rules to various named conventions such as Pascal/Camel Convention	Segments text according to rules
Abbreviation Expansion	Expansion rules and abbreviation dictionary for different domain	Expansion with dictionary and some heuristics algorithm.
Prefix/Suffix Strip	Configure the rules to strip prefix and suffix.	Strip prefix / suffix according to rules

B. Customization for Text Preprocessing: With the preprocessing library, it is unnecessary to implement preprocessing algorithms from scratch for handling different kinds of input. A group of preprocessing algorithms can be selected from library to handle different input. A text preprocessing container can be created to accommodate and run the preprocessing algorithms to handle the input text according to the configuration. Fig. 8 shows that once the

preprocessing container is configured appropriate it can output the text of natural language form.

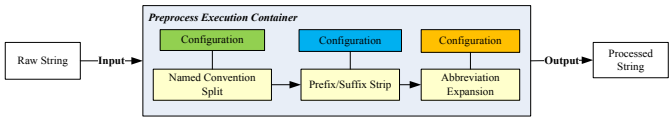

Fig. 8 Text Preprocessing Model

3.3. *Natural Language Processing*

Since corpora usually contain much natural language based information, it must be some relationships can be found out between Semantic Templates and corpora. Natural language processing (NLP) is such a kind of technology which can be used to exploit the relationship. Hence, NLP is one of the most important steps to generate high quality Semantic Template instance.

A. Syntax Analysis: In the natural language text, each word can be tagged by the part of speech such as verb, noun. Phrases and clauses can also be tagged such as verb phrase, noun phrase according to the syntactic element. For the Action Template, the verb property of the template may originate from a verb in the natural language text; target may originate from some noun phrase. From this case, it implies some relationship can be built between the syntactic elements of natural language and the properties of Semantic Template.

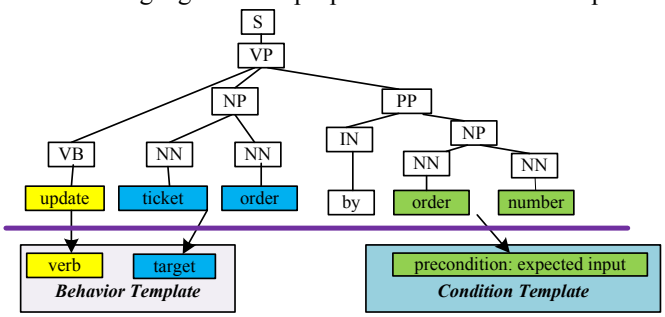

Fig. 9 Mapping Syntax Tree to Semantic Templates

Penn Treebank is a project defines a set of tags to annotate the natural language text. At word-level it can tag every word in a sentence with POS Tagset[6] (e.g. NN (Noun, singular or mass), VB (Verb, base form)), and at phrases or clause levels, a group of Syntactic Tagset are also available for tagging (e.g. S (simple declarative clause), NP (Noun Phrase)). The method to identify all of the syntactical elements over the natural language text and attach corresponding tags is called Bracketing[5]. We begin with a case, to pick up an operation name "UpdateTicketOrderByOrderNo" from some WSDL, and assume it will pass the Text Preprocessing first, and then using Penn Treebank II Tags annotate it. Fig. 9 shows the syntax tree for "update ticket order by order number", the leaf nodes correspond to the natural language text, and the ancestor nodes represent syntactical elements for different levels. From this figure we can find some relationship could be established between syntax tree and Semantic Templates. It makes possible to find a way to map phrase to some properties of Semantic Template schema.

B. Customization Patterns for Extraction: As the previous section analyzed, the verb and target property of Action Template respectively originate from VB and NP of operation name, and the NP dominated by PP can generate a part of precondition for Condition Template to ensure expected inputs. However, it probably the tags such as NP, PP are not unique in a syntax tree, so it is not a kind of one-to-one mapping relationship between a tag and a semantic property. A semantic property can map to one kind of tag, but in reverse direction, one kind of Tag, such as the NP nodes shown in Fig.9, which are possible mapping to different semantic properties of Semantic Templates. Following we will exploit a way to find the exact mapping relationship between phrase and semantic properties of Semantic Template.

The natural language sentences are always built on certain syntax structure. Once the syntax structure determined, the syntactical elements of the phrases are also determined. Software products are kinds of artifact which often contains rich natural language features. There are always some criteria and conventions for creating software products. For example, the operation name of Web Services usually conforms to the pattern of "verb+noun", this is called naming pattern. The operation name "UpdateTicketOrderByOrderNo" matches such pattern while "StudentIdToStudentName" matches another pattern: "NP to NP". These 2 patterns have different mapping relationships with Action Template. This show there are usually some commonly used patterns in software products, once the pattern is identified, the rule mapping from syntactical elements to Semantic Template can be determined.

Tregex[4] is a kind of tree pattern expression, which implements and extends Tgrep2[2]. It can be used for matching patterns in syntax tree. Hence, it is possible to extract semantic information from syntax tree by patterns. Base on above analysis, we have designed an approach to extract semantic information from the natural language text in 2 stages. The first stage is to identify the syntactic pattern of the input; the second stage is to extract phrases by matching the phrase patterns.

Stage1: Identifying Syntactic Pattern

To identify syntactic pattern from input, a group of checking rules can be defined for each syntactic pattern to check whether the input can match this pattern. It is to say if the input can match all the checking rules of certain syntactic patterns we could say this input belongs mathes this pattern. The formal description could be defined: If exists a set of syntactic patterns $\mathbf{P=\{P_1,P_2,\ldots,P_n\}}$ and each P_i have the checking rules $\mathbf{R_{Pi}=\{R_1,R_2,\ldots,R_n\}}$, then

$$(\exists \mathbf{P_i} \in \mathbf{P})\left(\forall \mathbf{R_j} \in \mathbf{R_{Pi}} \wedge \mathbf{T} \in \mathbf{R_j}\right) \Rightarrow \mathbf{T} \in \mathbf{P_i}$$

For example, the checking rule-set containing one rule for "verb+noun" pattern can be defined as: "**^VB/ >>, (__ !> __) $,, NP**", which can check if input text can match the pattern: a phrase beginning with a verb and a noun phrase following this verb. So the input "Update Ticket Order by Order Number" will match, while "Student Id to Student Name" will not.

Stage2: Extracting Phrases by Patterns

Once the syntax pattern of the input can be identified, the syntactic structure of the input is also determined. A couple of Tregex expressions can be created to locate the node in the syntax tree by matching defined phrase patterns. The leaves of

this node are the expected phrase to be extracted.

Table 2 Tregex Expressions for Extracting Phrases

Phrases	Patterns	Comments
P_{verb}	/^VB/ >>, (__ !> __)	Match the most left side verb node under root
P_{target}	NP !>> NP $,, (/^VB/ >>, (__ !> __))	Match the noun phrase which is the right sister of the above node
$P_{expectedInput}$	NP >> PP	Match the noun phrase dominated by prepositional phrase

Table 2 defines a couple of Tregex rules to extract phrases for {verb, target, expectedInput} from the input for the pattern of "verb + noun". For example, take "Update Ticket Order By Order Number" as the input, the output will be: {***verb*** = update, ***target*** = ticket order, ***expectedInput*** = order number}

3.4. Matching Referred Ontology

After NLP, a set of words or phrases mapping to the properties of Semantic Template has been extracted. The key problem has transformed to find the most matched ontology for these words or phrases. In the whole matching process, similarity algorithms, similarity score measure strategies and matching strategies should be considered, and an appropriate combination can be found by balancing the time and quality. The ontology has the maximum similarity score with the natural language phrase can be the good candidate referred by the semantic properties of Semantic Template instances.

A. Similarity Algorithms: In order to find out the most matched ontology, it should provide a lot of similarity comparison methods for matching from different perspective, such as Lexical Comparison (e.g. Levenstein, Ngram), WordNet Comparison (e.g. Lin, Jiang, Resnik). A similarity library can be funded to accommodate all of the commonly used similarity algorithms. With this library, various similarity algorithms can be selected to handle different types of inputs instead of create them every time.

B. Score Measure Strategies: It is with contingency to calculate the similarity score by only choosing one kind of similarity function. To eliminate such contingency, a group of various similarity functions should be selected to calculate the similarity scores respectively. Some measure strategy should be provided to deal with these scores and return a final similarity score. Following lists some score measure strategies which are used to handle the score set: Scores={S_1,S_2, …,S_n},

1. $\text{AverageScore} = \frac{\sum_{i=1}^{n} \text{Score [i]}}{|\text{Score }|}$
   where, |Score| is the length of the set, AverageScore is average value of the score set
2. $\text{MaxScore} = \text{Max}(\text{Score}[i])$
   where, MaxScore is the max similarity in the score set
3. $\text{WeightScore} = \frac{\sum_{i=1}^{n} W_i * \text{Score [i]}_i}{\sum_{i=1}^{n} W_i}$
   where, W_i is the weight for each similarity score, WeightScore is the weight based similarity score.

C. Phrase Matching Strategies: A phrase consists of a set of words, so it can be matched in 2 levels: 1) Chunk Level, the phrase will be taken as a whole string for comparison. 2) Word Level, the phrase will be taken as a set of words and it will be compared as a word set. Word Level matching is complementary with Chunk Level matching. For example, there are 2 phrases "CS Department" and "Department of CS", Chunk Level matching is probably returning a low similarity, but Word Level matching will take them as 2 word sets return a high similarity.

For Word Level Matching, the phrase for matching can be split as a word set: $\text{WordSet}(w_r)=\{w_1,w_2, \dots ,w_n\}$, and the certain label of ontology for matching can be taken as $\text{WordSet}(o_t)=\{o_1,o_2, \dots ,o_m\}$, these 2 word sets can be treated as a bipartite graph, then the maximum matching can be found in the bipartite graph. That is the best matching of words in these 2 word sets, the similarity of the phrase can be got. For example, a demonstration method shows bellow:

1. $\text{Sim}[m][n] = \text{Sn}\big(\text{WordSet}(w_r) \times \text{WordSet}(o_t)\big)$
   where, Sn represents some of the similarity algorithm, Sim[m][n] is the similarity Cartesian product of 2 word sets
2. $\text{MaxScore} = F_{\text{Max Matching}}\ (\text{Sim}[m][n])$
   where, $F_{\text{MaxMatching}}$ represents the maximum matching function
3. $\text{Similarity} = \frac{2*\text{MaxScore}}{|\text{WordSet }(w_r)|+|\text{WordSet }(o_t)|}$
   where, |WordSet| is the length of the word set

When to use Chunk Level or when to use Word Level Matching, it depends on the inputs and actual situation. Even these 2 matching strategies can be combined use if required.

D. Customization for Matching: To enable a flexible way to combine similarity algorithms, score measure strategies and matching strategies for similarity matching and a feasible way to take multiple data source as inputs regardless it is plain text, structure or semi-structure data, so it need build a universal flexible framework to achieve this goal. Our approach to achieve this goal is through extending SPARQL[1], because SPARQL is widely used as a general ontology query language in semantics world. One of the greatest benefits is it is unnecessary to modify the logics in program to handle different corpora or ontology models; instead the only thing need do is to update the SPARQL statements.

```
PREFIX sjtu: <java:cn.edu.sjtu.query.property.>
PREFIX matcher: <java:cn.edu.sjtu.similarity.matcher.>
PREFIX measurestring: <java:cn.edu.sjtu.similarity.measure.string.>
PREFIX measure: <java:cn.edu.sjtu.similarity.measure.strategy.>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?ontoref ?simscore
{
GRAPH ?graph
{ ?ontoref rdfs:label ?rdfslabel .}

?input sjtu:lowercase ?varWord .
?cmp sjtu:lowercase ?rdfslabel .
?leven sjtu:instance measurestring:Levenstein .
?lin sjtu:instance measurestring:LinSimilarity .
?resnik sjtu:instance measurestring:ResnikSimilarity .

?mscore sjtu:instance measure:MaxScoreMeasure;
  sjtu:arguments (?leven ?lin ?resnik) .

?m sjtu:instance matcher:MatchingAvgMatcher .
?m sjtu:arguments (?mscore ?input ?cmp) .
?m sjtu:score ?simscore .
FILTER (?simscore >= 0.7)}
ORDER BY DESC(?simscore) LIMIT 1
```

Similarity Algorithms

Score Strategy

Matching Strategy

Threshold Setting

Fig. 10 SPARQL Extensions for Matching Ontology

Fig. 10 shows the method that defines Similarity Algorithm, Score Strategy and Matching Strategy as a part of SPARQL statement. The natural language phrase can be bound to the input variable, and then run this configured statement to retrieve the most matched ontology from ontology models.

Once the ontologies for semantic properties are retrieved, they can be filled into Semantic Template instance according to the schema. The Semantic Template instances need be persistent for further use. It can be saved as a file or persisted into ontology database[21].

IV. Experiment and Evaluation

In the prototype system, we take WSDL files as the corpora for generating the instances of Action Template, Condition Template and Data Template automatically. As show in Fig.11, our system will generate SAWSDL through annotating the WSDL with these Semantic Template instances

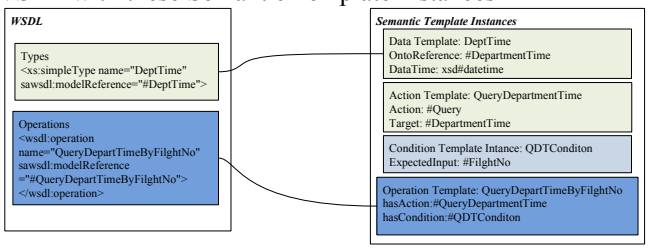

Fig. 11 Annotating WSDL using Semantic Template instances

In this experiment, we obtained the WSDL files and domain ontology files from Interne. Theoretically, it should have a complete domain ontology base which can cover all the concepts for each domain, but it impossible for us to create them in this experiment. When the prototype system generates the Semantic Template instances for some domain, it will merged domain ontology files into a whole ontology model to improve the recall ratio.

4.1. Resource for Experiment

Table 3 WSDL and Ontologies for experiment

Domain	WSDLs Files	Operations of WSDLs	Ontology Files	Ontology Files Size
Travel	7	62	7	140KB
Weather	9	66	4	1260KB
Publication	11	116	14	393KB
Finance	20	396	10	486KB

As shown in table 3, we categorize the WSDL files and domain ontologies into 4 domains: travel, weather, publication and finance. The ontologies for travel and weather are frequently mentioned in many papers about Semantic Web for experiments, so they are optimized and very stable. Actually, the ontologies for publication are also very widely used, such as Dublin Core. The ontology files for Finance is a little trivial, we can't find a kind of well-known ontology for this domain. WordNet Verb Ontology is also provided to generate verb semantic property for Action Template instances, which is extracted from [20] and it is domain independent.

4.2. Evaluation

We ran the test on each domain respectively, and evaluated the result by Recall and Precision. The Recall and Precision measures are obtained as follows:

$$\text{Recall}_X = \frac{RT_X}{RA_X},\ \text{Precision}_X = \frac{RC_X}{RT_X}$$

Where, $\boldsymbol{RA_X}$ is the set of all semantic property X should be retrieved. $\boldsymbol{RT_X}$ is actually the set of semantic property X retrieved. $\boldsymbol{RC_X}$ is the set of correctly retrieved semantic property X.

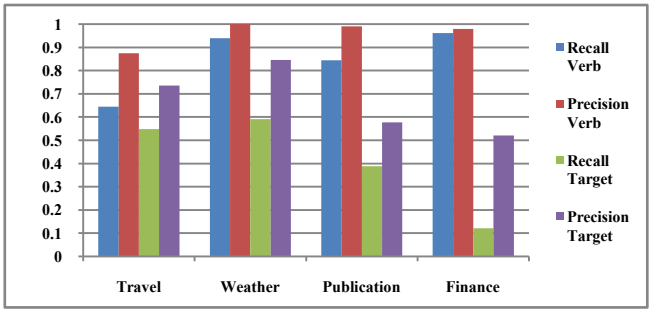

Fig. 12 Recall and Precision for Action Template

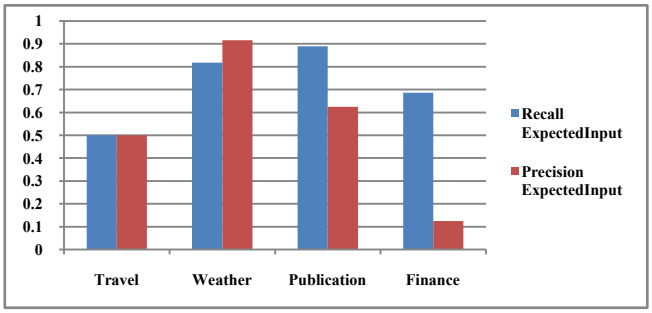

Fig. 13 Recall and Precision for Condition Template

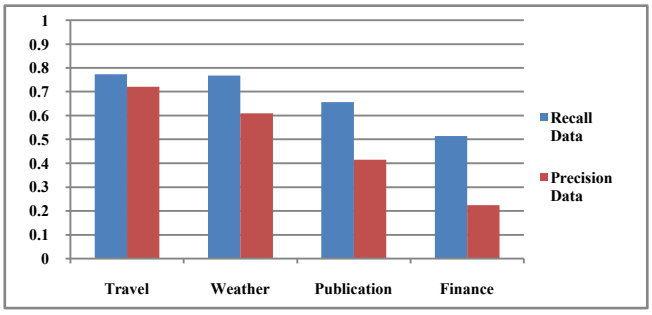

Fig. 14 Recall and Precision for Data Template

Fig. 12 depicts the recall and precision for generating the Action Template instances. The precision of "Verb" is very close to 1, because it matches ontology from WordNet Verb Ontology which is almost a complete ontology contains all the concepts of verb. "Target" is highly related to the domain concepts. Due to the incompleteness of the domain ontology for Finance domain, the recall of "Target" is only a little above 10% while benefitting from relative complete domain ontology the recall and precision of "Target" in Travel and Weather domain are above 60%.

Fig. 13 depicts the recall and precision for generating the "ExpectedInput" property of the Condition Template instances. In this experiment, this property is extracted from the preposition phrase in operation name. To a certain extent, the high recall proves the feasibility of extracting the phrases from natural language input by patterns. NLP can extract most of phrases with patterns, but if there is no sufficient

domain ontology such as in Finance domain, it still cannot map to the correct ontology, so the precision is very low.

Fig. 14 depicts the recall and precision for generating the Data Template instances. Data Template instances is highly domain concepts relevant, the more completeness of the domain ontology the higher precision will be, so the recall and precision for the Finance domain got the lowest rank again with the precision around 20%.

From this experiment, we can conclude the complete extent of the ontology has the direct impact on generating Semantic Template instances. It is impossible to present the detail of every step and configurations for our experiment in this section but the results prove that it is a feasible way to automatically generate most types of Semantic Templates.

V. Related Work

The well-known frameworks related to our work for Semantic Web Services, including WSMO[10], OWL-S[7], WSDL-S[11], SWSF[8]. WSMO is based on WSMF[3], which aims to create ontologies for describing various aspects related to SWS, with a more defined focus: solving the integration problem[14]. WSML[9][19] is a language takes into account all aspects identified by WSMO. WSML comprises different formalisms, most notably Description Logics and Logic Programming, in order to investigate their applicability in the context of ontologies and Web Services. OWL-S was the first major ontology for SWS, which defining a set of basic classes and properties for describing Web Services to enable agents to automatically discover, invoke and compose. SWSF, which includes two major components: SWSL[16] and SWSO[17], was devised to provide a full conceptual model and language expressive enough to describe the process model of Web Services, and to address the shortcomings of OWL-S in this regard. WSDL-S was created in the METEOR-S[18] project for annotating WSDL with semantic information. SAWSDL[15] is derived from WSDL-S, it does not provide a concrete model for SWS, instead it makes the assumption that the concrete model will be expressible as annotations in WSDL. There are already some works on OWL-S using SAWSDL for grounding[12][13].

In our approach, we propose to model Semantic Web Service using Semantic Template. However, we do not restrict the language to build Semantic Template. Besides RDF/S, OWL, both WSML and SWSL are rich in logical expressions, which can be good candidates to construct Semantic Template. Since it is able to construct a composite Semantic Template to describe any aspect of Web Service, it is very suitable to grounding the Semantic Template instance into SAWSDL to enable agent can locate it from "modelReference" for reasoning. Semantic Template also support modeling composite process more flexible than OWL-S.

VI. Conclusion and Future Work

In this paper, we proposed the Semantic Template for modeling Semantic Web Service. Base on the features of Semantic Template, it provides an effective way to model any aspect of Web Service avoiding the redundancy and inconsistency. We constructed some types of basic and advanced Semantic Templates to model some aspects of Web Service to enable agent intelligent discovery, composition and invocation. We also construct a flexible framework for automatic generating Semantic Template instance. Finally, the experiment proves that it is feasible to build various types of Semantic Templates for modeling Semantic Web Services.

In our recent work, we took WSDL as the corpora and it limited the types of Semantic Templates can be automatically generated. In our future work, we plan to support most of UML artifacts and other types of documents as the corpora. This will enable generate more complex Semantic Templates. Further, we plan to build a whole framework base on Semantic Templates to enable the all the functions of annotation, discovery and composition.

Acknowledgement

This work was supported by China Basic Research Project (under Grant No. 2003CB317005), and the National High-Tech Research and Development Plan of China (under Grant No. 2006AA04Z152, 2007AA01Z137).

References

[1] SPARQL Query Language for RDF, Available at http://www.w3.org/TR/rdf-sparql-query/, 2008

[2] Douglas L. T. Rohde , *TGrep2 User Manual*, 2005

[3] Dieter Fensel, Cristoph Bussler, *The Web Service Modeling Framework WSMF*, 2002

[4] Roger Levy, Galen Andrew, *Tregex and Tsurgeon: Tools for Querying and Manipulating Tree Data Structures*, 2006

[5] Ann Bies, et al., *Bracketing Guidelines for Treebank II Style Penn Treebank Project*, 1995

[6] Beatrice Santorini, *Part-of-Speech Tagging Guidelines for the Penn Treebank Project*, 1990

[7] The OWL Services Coalition. OWL-S 1.1. Available at http://www.daml.org/services/owl-s/1.1/, 2004

[8] Semantic Web Services Framework. SWSF Version 1.0. Available from: http://www.daml.org/services/swsf/1.0/, 2005

[9] J. de Bruijn, et al., *The Web Service Modeling Language WSML*, 2005, Available at http://www.wsmo.org/TR/d16/d16.1/v0.21/

[10] Roman, D., et al., *Web Service Modeling Ontology*, Applied Ontology 1(1), 77–106 (2005)

[11] Akkiraju, R., et al., *Web Service Semantics - WSDL-S, Tech. rep.*, LSDIS Lab. (2005), http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/

[12] Martin, D., Paolucci, M., Wagner, M., *Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective*, 2007

[13] Paolucci, M., Wagner, M., Martin, D., *Grounding OWL-S in SAWSDL*, 2007

[14] Rubén Lara, Dumitru Roman, Axel Polleres, Dieter Fensel, *A Conceptual Comparison of WSMO and OWL-S*, 2004

[15] Semantic Annotations for WSDL and XML Schema, Available at http://www.w3.org/TR/sawsdl/, 2007

[16] Semantic Web Services Language (SWSL), Available at http://www.w3.org/Submission/SWSF-SWSL/, 2005

[17] Semantic Web Services Ontology (SWSO), Available at http://www.w3.org/Submission/SWSF-SWSO/, 2005

[18] K. Verma, R.A., J. Miller, A. Sheth, *METEOR-S - An Environment for creating Semantic Web Processes*, VLDB Journal, 2004

[19] Dumitru Roman, et al., *WSMO, WSML, and WSMX in a Nutshell*, 2006

[20] RDF/OWL Representation of WordNet, Available at http://www.w3.org/2001/sw/BestPractices/WNET/wn-conversion.html, 2006

[21] H.Zhuge, et al., *Resource Space Model, OWL and Database: Mapping and Integration*, ACM Transactions on Internet Technology, 2008