Bootstrapping Operation-Level Web Service Ontology: A bottom-up Approach

Xumin Liu
Department of Computer Science
Rochester Institute of Technology
Email: xl@cs.rit.edu

Hua Liu
Xerox Research Center at Webster
Xerox Corporation
Email: hua.liu@xerox.com

***Abstract*—Ontology is the key ingredient of semantic Web service technologies, which support systematic management of Web services, such as automatic service discovery, service composition, and change management. It is crucial and challenging to reduce the human efforts for developing ontologies. We propose a bottom-up approach that bootstraps operation-level service ontologies from WSDL descriptions. The approach leverages the techniques of information retrieval and machine learning. The relevance and similarity between Web services are measured based on the WSDL descriptions. The process of developing service ontologies consists of two steps. First, we build service ontologies based on the service relevance. We then construct the structure of the service ontologies based on the service similarity. We conduct an empirical study on real Web services to demonstrate the effectiveness of the proposed approach.**

***Index Terms*—Web services, Ontology, WSDL**

I. Introduction

The last few years have witnessed a plethora of activities around Service-Oriented Computing (SOC). Many companies have already started delivering their business functionalities on the Web via Web services. Examples of these companies include Amazon, Ebay, Facebook, Force.com, Google, HP, IBM, Microsoft, and Yahoo. Meanwhile, the emergence and popularity of cloud computing further impel the rapid growth of Web services [1]. SOC has attracted considerable interests from both academia and industry. Many research efforts have been conducted aiming to make a full usage of Web services. A Web Service Management System (WSMS) has been envisioned that treats Web services as the first-class objects and manages them as how a DBMS manages data [16].

Automation is one of the most important and challenging research issues of Web service management. More specifically, due to the large scale and heterogeneous Web service space, realizing the full potential of Web services lies in the minimization of human efforts for the usage of Web services, such as service discovery, composition, and invocation. Semantic support is always considered as the key enabler for automation. The key idea of current semantic Web service technologies is to add machine-understandable semantics to service descriptions so as to allow software agents to capture the important information of a Web service and take over the work.

Ontology is the key ingredient of current semantic Web service technologies [4], [14]. The concept of ontology originally came from the field of Artificial Intelligence for recognizing objects and reasoning their relationships within a domain. Later it has been adopted in many other fields, including semantic Web services, to develop machine-understandable knowledge. There are two types of ontologies that have been proposed so far: *domain* ontology and *service* ontology. The domain ontology mainly focuses on data-level semantics. It describes concepts and their relationships in a certain domain so as to add semantic markups to the terms (e.g., input and output parameters) in a service description. The domain ontology allows automatic matching between user requests and service descriptions as well as improves the precision and recall of the matching result. The limitation of domain ontology is that the concept granularity is data in service descriptions. It does not directly capture the semantics of higher level objects, such as services and their relationships. The service ontology, on the other hand, focuses on service-level semantics. It treats services as the first-class objects when describing them and their relationships [7], [9]. The service ontology builds up a meaningful organization of Web services. The services having similar functionality are classified into the same categories, a.k.a., *service communities*. This organization gives a high-level and structured view of the important features of Web services, such as their functionality and inter-service relationships. Therefore, the service ontology allows a top-down, disciplined way to discover and compose Web services on a large scale [18].

The development and deployment of service ontologies have been seriously hindered by the intensive human efforts required during the process of ontology construction. Traditional approaches for ontology construction mainly rely on domain experts who have a comprehensive and thorough view on the concepts within a domain. It will be a very extensive, demanding, and even impossible work in the context of Web services. First, there is a large number of Web services available on the Web. The number still keeps increasing. Second, these Web services are offered by independent and autonomous service providers. Most of these providers only publish the WSDL descriptions of their services. However, it is not practical to go through all the WSDL documents by domain experts. A systematic support is needed to minimize the human efforts during the process of ontology construction.

Some research efforts have been conducted for automatically bootstrapping domain ontology [15], [12]. The proposed

COLLABORATECOM 2011, October 15-18, Orlando, United States

DOI 10.4108/icst.collaboratecom.2011.247159

approaches start with a set of Web sites that are coupled with Web services or WSDL documents. Machine learning algorithms and information retrieval approaches are applied to explore potential concepts and reason their relationships. Inspired by these approaches, we propose an approach that automatically extracts semantics from WSDL descriptions and constructs service ontologies. The approach starts with WSDL documents and identifies the functional features of Web services. It then builds up a hierarchical structure of service functionalities to form a service ontology. A Web service provides its functionality via a set of operations, which are unordered and unrelated to each other in a WSDL document. Therefore, it will be more suitable to use service operations as the granularity level to define basic functionalities than using services.

This paper is an extension of our previous work [8]. We present a two-phase process to construct service ontologies by mining service operations based on WSDL descriptions. During the first phase, we measure the functional relevance between service operations and use it to define service ontologies on a high level. Based on the relevance, we classify operations into functionally related groups. Each group corresponds to an application domain. Within a domain, we further study the similarity between service operations in the second phase. We use a hierarchical clustering algorithm to mine service operations based on their similarity. This process extracts common functional features of similar operations, which can be used to define abstract operations. The result of this process is a hierarchical structure of the service ontology. This service ontology reveals the relationship among services and operations, which is not directly visible from WSDL files. It allows to perform service discovery and composition in a structured manner.

The remainder of this paper is organized as follows. In Section II, we give an overview of the proposed process for building operation-level ontologies for Web services. We lay out and describe the key steps in the process. In Section III, we present a two-phase process that measures relevance and similarity, respectively, to mine service operations and construct service ontologies based on WSDL descriptions. In Section IV, we present a comprehensive experimental study to illustrate the effectiveness of the proposed approach. In Section V, we discuss some representative related work. We conclude our paper and discuss future work in Section VI.

II. An Overview of Ontology Development Process

In this section, we present the process of developing operation-level service ontologies. This process takes WSDL documents as input and generates a set of hierarchical service ontologies as output. As depicted in Figure 1, the process consists of several key steps, including *extract operations*, *compute operation relevance*, *build service ontology*, *compute operation similarity*, and *develop internal structure of ontology*. We elaborate on these steps as follows.

A. Extract Operations

The first step of service ontology construction is to extract the descriptions of operations from WSDL documents. A Web service provides its functionality via a set of operations. This makes operations the right granularity level to define basic functionalities of Web services. To analyze the functional features of a service operation, it is important to get as much information about the operation as possible from the WSDL document. During this step, we parse WSDL documents and store all the detailed description for each operation, including the operation name and detailed description of input and output messages. The message description includes the message name and the description of each part, which consists of the part name and its data type. Besides, we use the Web service and the service interface as the context of an operation to improve the accuracy of the analysis result. For this reason, service name and portType name are included in the operation's description. The output of this step is the descriptions of operations, which are stored in an *operation container*.

B. Compute Operation Relevance

This step is to compute the relevance between operations. The relevance between two operations measures how much they are related to each other. Generally speaking, two operations in the same application domain have a higher relevance than the ones in the different domains (e.g., travel, medical, and finance). For example, `flight_reservation` and `find_hotel` are expected to have a higher relevance than `flight_reservation` and `get_Medicine_name`. Although WSDL mainly describes a service at the syntactic level, information retrieval techniques can be adopted to extract semantics from WSDL descriptions. This is due to the observation that some common naming conventions are usually followed for Web service development, especially for the WSDL documents which are automatically generated from programming source codes. For example, an operation usually has the name of the original function, such as `TemperatureConversion`. Based on this observation, we can analyze the functional features of a service operation from the terms in its description. Following these lines, we extract *terms* from an operation's description to compute the operation relevance.

It is very common that an element in a WSDL document appears in a composite format. For example, an operation may have a name like `get_Map`, `sendPurchaseRequest`, or `order1`. Thus, tokenization is performed on an operation's description to extract simple terms. The tokenization process decomposes a given expression into simple terms. It consists of *case change*, *suffix numbers elimination*, *word stemming*, and *underscore separator* [10]. The output of the tokenization process is a set of terms that are used to describe a service operation.

The relevance of two operations is computed based on their terms. We elaborate on the computation in Section III-A. The operation relevance matrix that stores the relevance between any pair of operations is generated as the output of this step.

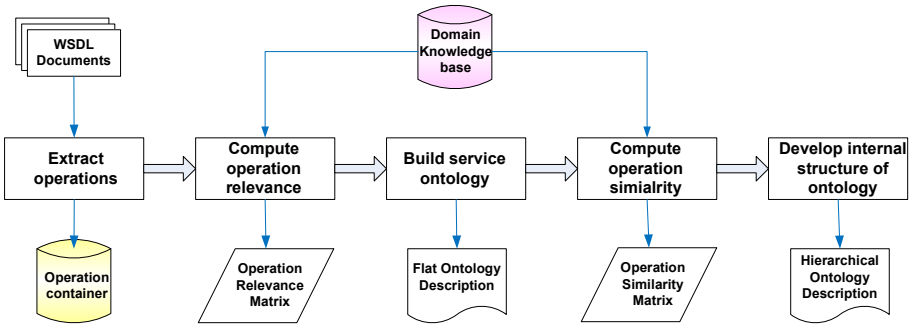

Fig. 1. The Ontology Construction Framework

C. Build Service Ontology

This step takes the operation relevance matrix as input and identifies different domains and the corresponding services in each domain. A service ontology is then created for each domain. This step uses the matrix to cluster the service operations by grouping the related operations together. It is elaborated in Section III-A.

D. Compute Operation Similarity

This step is to compute the similarity between two relevant operations, i.e., the operations in the same domain. It is different from measuring relevance between operations, where only terms are considered. When computing the similarity, the structure of an operation (e.g., input and output messages) is also considered. The reason is that two similar operations should have both similar syntactic features (i.e., names) and structures, which need to be evaluated accordingly. For example, `get_map` and `get_route` are expected to have a higher similarity than `get_map` and `flight_reservation`. The output of this step is the operation similarity matrix, which stores the similarity between any pair of operations in a service ontology. This step is elaborated in Section III-B.

We use a *domain knowledge base* to improve the accuracy of computing service relevance and similarity. The knowledge base describes a set of terms and their relationships. The usage of the domain knowledge base is due to the syntactic difference between synonyms. For example, although `trip` and `journey` share similar meaning, they will be treated as two distinct terms. Wordnet [1], a lexical database, has been widely used to connect between synonyms. Another option is to rely on Web search engines to compute the distance between two terms [11]. Since Wordnet has been demonstrated to be effective to retrieve synonyms for a term, we choose to use Wordnet in our work.

E. Construct Internal Structure of Ontology

This step takes the operation similarity matrix as input and generates the internal structure for each service ontology. A hierarchical clustering approach is performed to cluster similar operations and extract their common features. This process is elaborated in Section III-B.

[1]http://wordnet.princeton.edu/

III. Ontology Construction

In this section, we present the process of constructing operation-level ontologies. It follows two steps. First, we create a service ontology for a certain domain by grouping operations providing related functionalities together. Second, we build up a hierarchical structure for the service ontology based on the functional similarity between service operations.

A. Relevance-based Ontology Construction

The process of relevance-based ontology construction consists of two steps. We first compute the relevance between service operations and generate the relevance matrix. We then define service ontology based on the relevance matrix.

The relevance between two operations is computed based on their descriptions (e.g., name, input message, and output message) and their context (Web service and portType). In information retrieval techniques, terms are typically used as the basic elements to compare between two documents. This idea can also be applied to operation comparison. It is due to the observation that the more *representative terms* are shared by the two operations, the higher relevance that these two operations should have. The representative term refers to the one that can represent the functionality of a service operation and it tends to have high appearance frenquency among service operations in the same domain. Therefore TF/IDF [2] can be used to measure the degree of representativeness of a term. To improve the precision and recall of the result, we use Wordnet to solve the syntactic difference between synonyms.

Suppose that $\mathcal{T}$ is the set of terms extracted from all WSDL documents. It contains *k* terms with distinct meaning. We can model an operation as a *k* dimensional term vector. Thus, giving *m* operations, we can generate an $m \times k$ operation-term matrix $\mathcal{M}^{\mathcal{OP}}$, where each row represents an operation and each column represents a term. More specifically, let t_j be the jth term and op_i be a set of terms that are included in the ith operation's description or context, we have:

$$\mathbf{m}^{\mathbf{OP}}_{\mathbf{i,j}} = \begin{cases} 0 & \text{if } t_j \not\in' \mathbf{op_i} \\ TF'(t_j)/IDF'(\mathbf{op_i}) & \text{otherwise} \end{cases} \quad (1)$$

$t_j \not\in' op_i$ means that t_j or its synonyms does not belong to op_i. We also treat a term and its synonyms as the same term

when computing its TF/IDF. After generating the operation-term matrix, we apply kmeans clustering to group related operations together. The reason of choosing kmeans is that it is very efficient with time complexity $\mathcal{O}(KIM)$ and it can achieve high accuracy of the clustering result.

The result of the clustering process is a set of operation clusters, where operations providing the related functionality are grouped to the same cluster. Each cluster corresponds to an application domain. A service ontology is generated from each cluster. The operations in the cluster constitute the content of the ontology.

B. Similarity-based Ontology Construction

The process of similarity-based ontology construction consists of two steps. First, we compute the similarity between two service operations within an ontology and generate the similarity matrix. We exploit a similar strategy as developed in [10] for measuring similarity between two service operations. Second, we build up the hierarchical structure of the ontology based on the similarity matrix.

The similar service operations refer to the ones that provide similar functionality, which can be described by their names, input messages, and output messages. Therefore, we compute the similarity between two operations by comparing them using these three parameters, respectively. Different weights can be assigned to them. More specifically, we define the similarity between two operations as:

$$\begin{aligned} simOP(op_i, op_j) = \\ & w_1 \times simName(op_i.name, op_j.name) + \\ & w_2 \times simMsg(op_i.in, op_j.in) + \\ & w_3 \times simMsg(op_i.out, op_j.out) \\ & \text{where } w_1 + w_2 + w_3 = 1. \end{aligned} \quad (2)$$

Therefore, the similarity comparison between two operations is decomposed to the comparison between names and the comparison between messages. We elaborate on these two comparisons as below.

It is typical that an operation has a composite name, such as get_Purchase_Order or carRentalReservation. Therefore, we need to first tokenize the operation name and decompose it to simple terms for comparison. Let T_{op_i} be the set of terms decomposed from $op_i.name$ and T_{op_j} be the set of terms decomposed from $op_j.name$,

$$simName(op_i.name, op_j.name) = simTermSet(T_{op_i}, T_{op_j}). \quad (3)$$

We model the two sets of terms as two sets of nodes in a bipartite graph and compute the maximum weight matching as their similairity. A bipartite graph $G = \{N_1, N_2, E\}$. N_1 and N_2 are two disjoint sets of nodes in G. E is a set of edges between the nodes in N_1 and the nodes in N_2. There are no edges between two nodes from the same set. A matching M is defined as a subset of E, where there are no two edges in M sharing a same end node. An edge in M is weighted. The weight is assigned by the term similarity function $simTerm(t_1, t_2) \rightarrow [0..1]$. We calculate the term similarity by leveraging the approach proposed in [3]. We first compute the normalized compression distance (NCD) between two terms. We then get the similarity as:

$$simTerm(t_1, t_2) = 1 - NCD(t_1, t_2) \quad (4)$$

As depicted in Figure 2, T_i and T_j are two sets of terms. The maximum weight matching is the one that has the maximum sum of weights of edges, i.e., $\{< t_{i1}, t_{j1} >, < t_{i2}, t_{j2} >\}$.

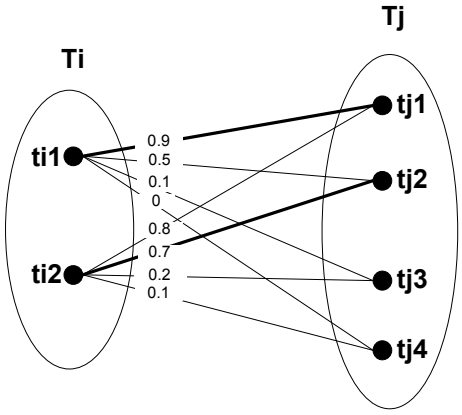

Fig. 2. The bipartite graph model of two term sets

The weight of a matching is:

$$W_M = \sum_{i=1}^{i=|M|} simTerm(e_i.from, e_i.to) \quad (5)$$

Suppose there are n matchings in G, i.e., $M_1, ..., M_n$. Therefore, the similarity between two sets of terms is defined as:

$$simTermSet(T_{op_i}, T_{op_j}) = \frac{\max_{1 \le k \le n}(W_{M_k})}{\min(|T_{op_i}|, |T_{op_j}|)}, \quad (6)$$

We use Equation 3 and 4 to compute the similarity between two message names. We compute the similarity between two messages by comparing their names and parts. Different weights can be assigned to these two aspects. More specifically, we define the similarity between two messages as:

$$\begin{aligned} simMsg(m_i, m_j) = \\ & w_1 \times simName(m_i.name, m_j.name) + \\ & w_2 \times simMsg(m_i.Parts, m_j.Parts) \\ & \text{where } w_1 + w_2 = 1. \end{aligned} \quad (7)$$

It is likely that two similar parts are defined with different data type granularity, which will affect the result of comparison. For example, the input message of a `Geocode` service has one part, `address`, which has a complex data type. Another service has four parts in its input message, including: `street_Info`, `city`, `state`, and `zip`. To better compute the similarity between parts, we first flatten each part to a set of *atomic parts*, where each atomic part has an atomic (or standard) data type. We then use the bipartite graph model to compute the similarity between the parts of two messages.

We calculate the maximum weight matching of the message part bipartite graph as the similarity. More specifically, we use $\mathcal{AP}_p$ to denote the flattened set of the part p. In the graph, $N_1 = \cup_{p \in m_i.Parts} \mathcal{AP}_p$, and $N_2 = \cup_{p \in m_j.Parts} \mathcal{AP}_p$. The weight assignment function between two atomic part $simPart(ap_1, ap_2) \rightarrow [0..1]$ is defined as:

$$simPart(ap_1, ap_2) = w_1 \times simName(ap_1.name, ap_2.name) + w_2 \times simType(ap_1.type, ap_j.type) \quad \text{where } w_1 + w_2 = 1. \tag{8}$$

Till now, we have explained the process of calculating the similarity between two operations. The similarity will be used to construct operation-level ontologies using data clustering algorithm

We choose to use SLINK algorithm [13] for ontology construction for two reasons. First, SLINK is a hierarchical clustering algorithm, which is suitable for constructing the hierarchical structure of service ontology. Second, different from other clustering algorithms, such as kmeans where "centroids" need to be computed for clustering, SLINK algorithm clusters objects based on the single-link distance, which is the minimum distance (i.e., maximum similarity) between any object in the first cluster and any object in the second cluster. This fits for clustering operations since it is not feasible to compute "operation centroids". Although SLINK algorithm is an expensive algorithm with time complexity $\mathcal{O}(M^2 lgM)$, it achieves high accuracy on the result. Considering that we only apply SLINK algorithm to cluster the operations in the same ontology, the overall process is still efficient.

The result of the hierarchical clustering is a multi-level cluster of service operations. Each cluster represents an abstract operation, which contains the common functional features of service operations grouped in the cluster.

IV. Experimental Study

We conducted a set of experiments to assess the effectiveness of the proposed ontology construction framework. We run our experiments on a Mac Pro with 2.66 GHz Quad-Core processor and 6GB DDR3 memory under Mac OS X operating system. The experiments are conducted on a set of real Web services provided by [2]. To clearly present the experimental results, we choose a set of representative services from each of the five application domains in the given Web service data set. More specifically, we include 19 medical services, 19 communication services, 18 food services, 20 travel services, and 26 education services.

A. Performance of Relevance-based Ontology Construction

The operation ontologies are created by first clustering service operations based on their relevance. The main objective of this step is to group together operations that offer related functionalities. Since the structure of the ontologies is not a concern at this point, most existing clustering algorithms can be used for this purpose. We choose the widely used k-means algorithm and set k as 5. Table I shows the operations that are assigned to each ontology. We compute the precision and recall based on the clustering results for each ontology. The precision and recall are calculated as follows:

$$\text{precision} = \frac{\text{the number of correctly clustered operations}}{\text{the total number of operations clustered into the ontology}} \tag{9}$$

$$\text{recall} = \frac{\text{the number of correctly clustered operations}}{\text{the total number of operations from the given domain}} \tag{10}$$

As can be seen, perfect precision and recall are achieved for three ontologies: communication, food, and education. The algorithm mis-clusters 8 travel operations into the medical ontology so the precision for the medical ontology is 70.4% and the recall of the travel ontology is 60%.

B. Performance of Ontology Structure Construction

We present the performance on ontology structure construction in this section. We choose to use a hierarchical clustering algorithm to further cluster the operations that are assigned to the same ontology. This allows us to directly construct the hierarchical structure of the service ontology. Comparing the ontologies generated by this algorithm and manual process, it reveals a high similarity. Limited by space, we will only show the resultant structure of the travel ontology. Other ontologies present a very similar structure.

Figure 3 shows the travel ontology structure obtained from hierarchical clustering of operations in the travel ontology. The bottom level (i.e., level 0) corresponds to the indices of all the operations that are assigned to this ontology. Table II gives the detailed information of each of these operations, including operation name, input message, and output message. Since the input and output of an operation may consist of multiple parameters, the name and data type of each parameter is also listed in the table. Some parameters have a complex data type. For example, _ORGANIZATION has a complex data type that includes multiple components, like the size, sub-unit, and headed by etc. We omit these information for the sake of space.

As can be seen in Figure 3, the clustering proceeds in a hierarchical fashion. For example, at level 1, the most similar operations are clustered into a set of very cohesive clusters. Operations in the same level-1 clusters provide identical or very similar functionalities. For example, operations 7 and 8 both are used to get surfing destination information for organizations. As the clustering process proceeds, more loosely coupled clusters are generated. For example, operations 7, 8, 0, and 4 are grouped together as a level-2 cluster. These operations are used to get the destination information for either surfing or hiking. Obviously, the hierarchical ontology structure captures more information than a flat ontology structure. It not only provides the similarity between different operations but also captures the similarity between different operation clusters. Such a structure also greatly facilitate us in identifying abstract operations which can be used to describe the functionalities of the operation ontologies. As an example,

[2]http://projects.semwebcentral.org/projects/owls-tc

TABLE I
RESULT OF RELEVANCE-BASED CLUSTERING

Ontology name	Operations	Precision	Recall
Communication ontology	19 communication operations	100%	100%
Food ontology	18 food operations	100%	100%
Medical ontology	19 medical operations 8 travel operations	70.4%	100%
Travel ontology	12 travel operations	100%	60%
Education ontology	26 education operations	100%	100%

we identify three abstract operations A_0, A_1, and A_2: A_0 can be regarded as a very generic travel related operation; A_1 can be regarded as an abstract operation that provides surfing and/or hiking related information; and A_2 can be regarded as an abstract operation that provides accommodation related information. If needed, more abstraction operations can be easily identified by just following the hierarchical ontology structure.

The ontology hierarchy enables easy service discovery and seamlessly dynamic service replacement. For example, if a user looks for services that provide destination information for either surfing or hiking, operations 7, 8, 0 and 4 are returned according to the ontology hierarchy. Higher accuracy can be achived using this ontology hierarchy based approach compared to keywords based search. As another example, operation 7 can be dynamically swapped with operation 0 without disruption to the workflow that operation 7 belongs to, when operation 7 becomes unavailable. This is because that operation 7 and 0 are clustered together at the same level of the hierarchy reveals their replaceability.

V. RELATED WORK

Our work is related to two topics: developing ontology for Web services and extracting semantics from WSDL documents. In this section, we discuss some representative related works and differentiate our approach from them.

A. Ontology Development

The approach proposed in [12] aims to automatically generate domain ontologies for Web services in a given domain. The ontological bootstrapping process takes WSDL descriptions and free text descriptions of Web services as input. It first extracts terms from a WSDL document and then computes the TF/IDF for each term. The terms having high TF/IDF will be selected. Meanwhile, it also performs Web context extraction to produce a set of context descriptors, which have the most references in number of Web pages and in number of appearances in the WSDL documents. The terms that appear in the result of both the TF/IDF computation and Web context extraction will be considered as potential concept names. The concept relationships can be generated by using context descriptors. The current ontology will be verified and evolved by the free text description of the Web service.

In [15], a system, "DeepMiner" is proposed to automatically derive domain ontologies for semantically marking up Web services. It takes a set of web sites that potentially provide Web services in a domain as input and uses machine learning approaches to incrementally learn domain ontologies. DeepMiner observes the query interfaces and data pages of the web sites. A base ontology is first generated from the query interfaces. DeepMiner then grows the ontology by investigating more information from the data pages. SLINK algorithm is used to discover distinctive concepts over multiple interfaces.

The approaches proposed in [12] and [15] mainly focus on domain ontologies, which capture the data-level concepts and their relationships. The resource used for deriving ontology consists of the query interfaces and data pages of web sites. Our work focuses on service ontologies, which treats services, more specifically, service operations as first-class objects and capture their common functional features and relationships.

B. Semantic Extraction from WSDL

In [17], a co-clustering approach is proposed to generate Web service communities based on WSDL descriptions. The approach improves the precision and recall of community generation by clustering Web services and operations together. It builds up a service matrix and an operation matrix based on their term TF/IDFs. The similarity between a Web service and an operation is computed as a dot product of the service vector and the operation vector. A co-occurrence matrix of services and operations is modeled as an undirected bipartite graph which consists a set of service nodes, a set of operation nodes, and the edges between them. Each edge is weighted as the similarity between the corresponding service and operation. Based on the bipartite graph model, the Singular Vector Decomposition (SVD) approach is used to group related Web services and operations into the same communities.

The work proposed in [5] applies a clustering algorithm, Quality Threshold (QT), to cluster Web services into functionally similar service groups. It measures the similarity between two services by comparing the elements in WSDL documents, including service names, complex data types, messages, portTypes, as well as terms.

The approaches in [17] and [5] generate *flat* service communities. In contrast, our approach generates a service ontology, which builds up a hierarchical structure on service functionalities. In addition, the approach proposed in [5] only considers two comparison results: matched or unmatched when comparing between two elements. This does not conform to the fact that two elements can also be partially matched. When comparing between two sets of terms, it computes the similarity between all pairs of terms from the two sets and averages the sum as the final result. However, the comparison

TABLE II
OPERATIONS IN THE TRAVEL ONTOLOGY

ID	Operation name	Input message	Output message
0	get_DESTINATION	[_SURFING, (xsd:string)]	[_DESTINATION, (xsd:string)]
1	get_NATIONALPARK	[_SURFING, (xsd:string)]	[_NATIONALPARK, (complex)]
2	get_CITY	[_GENERIC-AGENT, (xsd:string)], [_SURFING, (xsd:string)]	[_CITY, (complex)]
3	get_CITY	[_HIKING, (xsd:string)], [_SURFING, xsd:string)]	[_CITY, (complex)]
4	get_DESTINATION	[_HIKING, (xsd:string)], [_SURFING, (xsd:string)]	[_DESTINATION, (xsd:string)]
5	get_NATIONALPARK	[_HIKING, xsd:string)], [_SURFING, (xsd:string)]	[_NATIONALPARK, (complex)]
6	get_CITY	[_ORGANIZATION, (complex)], [_SURFING, (xsd:string)]	[_CITY, (complex)]
7	get_DESTINATION	[_ORGANIZATION, (complex)], [_SURFING, (xsd:string)]	[_DESTINATION, (xsd:string)]
8	get_DESTINATION	[_ORGANIZATION, (complex)], [_SURFING, (xsd:string)], [_PERSON, (complex)]	[_DESTINATION, (xsd:string)]
9	get_ACCOMMODATION	[_GEOPOLITICAL-ENTITY, (xsd:string)], [_TIME-MEASURE, (xsd:string)], [_CITY, (complex)]	[_ACCOMMODATION, (xsd:string)]
10	get_BEDANDBREAKFAST	[_GEOPOLITICAL-ENTITY, (xsd:string)], [_TIME-MEASURE, (xsd:string)], [_CITY, (complex)]	[_BEDANDBREAKFAST, (xsd:string)]
11	get_HOTEL	[_GEOPOLITICAL-ENTITY, (xsd:string)], [_TIME-MEASURE, (xsd:string)], [_CITY, (complex)]	[_HOTEL, (xsd:string)]

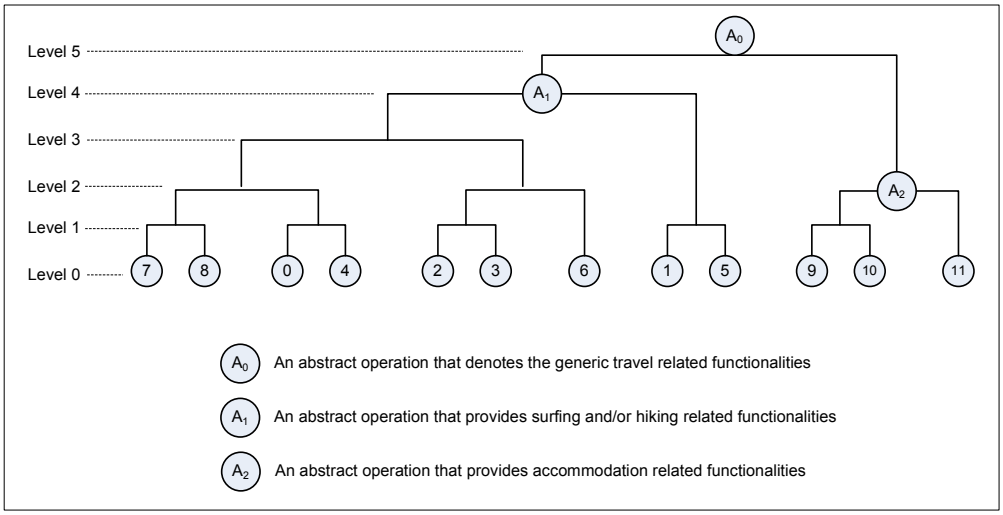

Fig. 3. The Hierarchical Travel Ontology Structure

between two unrelated terms may be meaningfulness. In our work, we look into the internal structure of elements (e.g., messages and parts) when measure community and accept partial matching. We also use bipartite graph model to improve the accuracy of similarity calculation.

In [10], URBE (Uddi Registry By Example) is proposed to intelligently retrieve Web services based on similarity between Web service interfaces. The similarity between two WSDL documents is computed based on the elements and the terms included in the documents. It defines a maximization function to calculate the similarity between the elements in two sets, based on a bipartite graph model. It then uses the maximization function to compute the similarity between names, operations, names, and parts. The work also utilizes Wordnet to solve the syntactic conflicts between synonyms. URBE is then extended to compute similarity between semantically annotated Web service descriptions, i.e., SAWSDL documents.

In [6], an approach for measuring similarity between two WSDL documents is proposed aiming to improve the work in [10]. The improvement has been made in two aspects. First, it uses Web search engines to compute the similarity between two terms, instead of relying on Wordnet, The purpose is to improve the accuracy and flexibility, as well as to capture the implied connections between two terms. Second, it proposes

two-phase similarity metrics to deal with unbalanced bipartite graphs. This is due to the observation that the unmatched terms in an unbalanced bipartite graph may have effects on the similarity, which is ignored in work [10]. The first phase similarity follows the same process proposed in [10]. In the second phase, the unmatched terms and their weights are considered to compute the similarity.

The work proposed in [10] and [6] computes the similarity between two WSDL interfaces to match user requests and service providers or find a substitute service. Our work focuses on constructing operation-based ontology for Web services, which defines a hierarchical structure for service functionalities for efficient usage of Web services. We consider both relevance and similarity when comparing service operations. Based on the relevance, we cluster related service together to define service ontologies and generate their contents (i.e., the service operations in a service ontology). Based on the similarity, we build up the internal structure of service ontologies.

VI. Conclusion

We present a bottom-up approach to bootstrap operation-level ontologies for Web services. The ontology construction is based on the calculation of relevance and similarity of service operations. We model service operations as term vectors and apply kmeans algorithm to efficiently measure their relevance. Service ontologies are formed and their contents are determined based on the relevance. We then compute the similarity between two operations in a service ontology and apply SLINK algorithm to build up the internal structure of the ontology. In the future work, we plan to apply our approach to semantic service descriptions, such as OWL-S, where more information (e.g., preconditions and effects) can be used to better construct service ontologies.

Acknowledgment

This work is supported by Xerox research grant.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53:50–58, April 2010.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] R. L. Cilibrasi and P. Vitanyi. The google similarity distance. *IEEE Trans. on Knowl. and Data Eng.*, 19:370–383, March 2007.

[4] The OWL Services Coalition. Owl-s: Semantic markup for web services. http://www.daml.org/services/owl-s/1.1B/owl-s/owl-s.html, July 2004.

[5] K. Elgazzar, A. E. Hassan, and P. Martin. Clustering wsdl documents to bootstrap the discovery of web services. In *ICWS 2010*, pages 147–154, 2010.

[6] F. Liu, Y. Shi, J. Yu, T. Wang, and J. Wu. Measuring similarity of web services based on wsdl. In *ICWS 2010*, pages 155–162, 2010.

[7] X. Liu, C. Liu, M. Rege, and A. Bouguettaya. Semantic support for adaptive long term composed services. In *ICWS 2010*, Miami, FL, July 2010.

[8] X. Liu and H. Liu. Constructing operation-level ontologies for web services. In *ICWS 2011 (Work-In-Progress)*, Washington DC, July 2011.

[9] B. Medjahed and A. Bouguettaya. A multilevel composability model for semantic web services. *IEEE Trans. on Knowl. and Data Eng.*, 17(7):954–968, 2005.

[10] P. Plebani and B. Pernici. URBE: Web service retrieval based on similarity evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21:1629–1642, 2009.

[11] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, 2006.

[12] A. Segev and Q. Z. Sheng. Bootstrapping ontologies for web services. *IEEE Transactions on Services Computing*, 99(PrePrints), 2010.

[13] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16:30–34, 1973.

[14] WSMO Working Group. Web Service Modeling Ontology (WSMO). *http://www.wsmo.org/*, 2004.

[15] W. Wu, A. Doan, C. Yu, and W. Meng. Bootstrapping domain ontology for semantic web services from source web sites. In *In Proceedings of the VLDB-05 Workshop on Technologies for E-Services*, pages 11–22, 2005.

[16] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed. Deploying and managing web services: issues, solutions, and directions. *The VLDB Journal*, 17:537–572, May 2008.

[17] Q. Yu and M. Rege. On service community learning: A co-clustering approach. In *ICWS 2010*, pages 283–290, 2010.

[18] Q. Yu, M. Rege, A. Bouguettaya, B. Medjahed, and M. Ouzzani. A two-phase framework for quality-aware web service selection. *Service Oriented Computing and Applications*, 4:63–79, 2010.