An Integrated Framework for Web Service Ontology Development

Xumin Liu
Rochester Institute of Technology
and
Hua Liu
Xerox Research Center at Webster Xerox Corporation

In this paper, we propose an integrated framework to construct a Web service ontology in a bottom-up fashion. The ontology treats Web services as the first-class objects, which captures their functional-features and inter-service relationships. The proposed framework starts with a set of WSDL service descriptions given that WSDL is the *de facto* standard for describing service APIs. Information retrieval techniques are leveraged to extract a Web service's functional features from its WSDL description. The framework measures the functional relevance between service operations and uses it to define the service ontology on a high-level. An improved data clustering algorithm is proposed to group relevant operations. Each operation group corresponds to a service domain. Within a domain, the framework measures the functional similarity between service operations and applies a hierarchical data clustering algorithm to construct the internal structure of the service ontology. A comprehensive experimental study is conducted to evaluate the efficiency and effectiveness of the framework.

Keywords: Web services, Ontology, WSDL

1. INTRODUCTION

The newly emerging service-oriented and cloud computing have gained great momentums from both academia and industry as they are considered as the next generation of computing paradigms. As a result, the variety and amount of Web-based services have been tremendously increased. Many research efforts have been conducted to make a full usage of these Web services, such as efficient discovering desirable services [Yu and Bouguettaya 2008], composing multiple services to provide value-added services [Medjahed et al. 2003], enforcing security and privacy protection of Web services [Rezgui et al. 2002], and managing evolutional and exceptional changes during the lifetime of service-oriented systems [Liu and Bouguettaya 2007; Akram et al. 2010]. A systematic support that treats Web services as the first-class objects and efficiently manages and manipulates them to realize their full potential has been envisioned [Yu et al. 2008]. To realize such a system, it is essential to minimize the human effort required for the usage of Web services.

Semantic Web service technologies aim at improving the automation by adding machine-understandable semantics to service description [Coalition 2004; WSMO Working Group 2004; W3C 2005]. The proposed Web service related ontology, as the kernel of semantic Web service technologies, falls into two categories: *domain ontology* and *service ontology*. Domain ontology describes term-level concepts and their relationships. It can be used to add semantic markups to a service's description. The input, output, and operations of a Web service are annotated with machine-understandable information, which helps software agents discover and invoke a Web service. Domain ontology has enjoyed extensive support from legacy efforts in the fields of semantic Web and knowledge engineering. However, it is limited to its conceptual granularity, which is on term level instead of service level. It does not capture the inter-service relationships and does very little to build up service organization, which is essential to support a systematic management of Web services. Service ontology, on the other hand, treats Web services as first class objects for conceptual modeling. It builds up a meaningful organization of Web services

Author's Email: xl@cs.rit.edu, hua.liu@xerox.com

based on their functional features. The services having the same functionality are grouped into the same categories, *a.k.a., service communities*. Beyond a group of *flat* service communities, a service ontology has a hierarchical structure, similar to a domain ontology, which captures inter-community relationships. This hierarchical structure of a service ontology defines service functionality at different levels, providing a complete and structured view on all available services.

The current approaches of developing and deploying a service ontology are mainly performed in a *top-down* fashion. In a top-down method, a service ontology is predefined by domain experts through a thorough study on a service space. Service providers, will relate their services to the service ontology when describing and publishing their services. There are two major shortcomings of top-down methods. First, intensive human efforts are required during the process of constructing service ontology. More specifically, domain experts need to go through the service descriptions of all available services in a service space to design the service ontology, which is obviously not suitable for a large-scale environment. It does not also cope with the ever increasing size of a service svvvvpace. Second, top-down methods assume that service providers will link their services to the service ontology. However, service providers are usually independent and autonomous. Most of them are reluctant to make such extra efforts. It is not practical to make this assumption then.

We propose a bottom-up method to address the shortcomings of top-down ontology development approaches. The main idea of the bottom-up method is that, it starts with service descriptions, which have already been published by service providers and available on the Web, and automatically extract inter-service relationships to construct service ontologies. Although many semantic Web service descriptions have been proposed, such as OWL-S [Coalition 2004], WSMO [WSMO Working Group 2004], and WSDL-S [W3C 2005], most service providers still use WSDL [W3C 2001], the *de facto* standard, to describe their services. Therefore, it is more practical to build our method based on WSDL descriptions than other descriptions. It is well known that WSDL provides limited, and mainly syntactic-level description of Web service functionality. To conquer this limitation, we leverage information retrieval and machine learning techniques, which have been widely used in developing domain ontology. Our approach first extracts functional features from WSDL descriptions, compares different services to determine their relationships, and builds up a hierarchical structure of service functionalities to form a service ontology. It is worth to note that a Web service performs its functionality via a set of operations, which are unordered and sometimes unrelated to each other in a WSDL document. The association between a service and the generated service ontology is derived during the process of ontology construction as the by productions. The bottom-up method minimizes the human efforts and achieves much better scalability than top-down methods. Therefore, our approach will use operation as the concept granularity when building service ontology. The major contributions of this paper are summarized as follows:

— We present an integrated framework that supports automatic service ontology development. This framework crawls the Web service space, retrieves WSDL descriptions of services, identifies different domains for these services, and generates a hierarchical service ontology for each domain. The ontology can be presented to domain experts to refine or end users to use. No human effort is required during this process. The key components of this framework are described.

— 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. In [Liu and Liu 2011], we applied K-means to cluster operations. K-means is a widely used data clustering technique which achieves high accuracy with efficient performance. However, it has two major limitations. First, the quality of clustering result is affected by the initial centroids, which are usually randomly selected. Second, we have to specify k's value to perform K-means algorithm, which limits the applicability of K-means. In this paper, we leverage Bisecting K-means and cluster

quality evaluation approaches to address the two limitations simultaneously.

— Within a domain, we study the similarity between service operations. This helps improve the efficiency of the overall process by avoiding of similarity comparisons between unrelated services. We use a hierarchical clustering algorithm to mine service operations based on their similarity. This process extracts common functional features of similar operations, which presents the functionality abstraction at different levels that can be used to define abstract operations. The abstract operations and the hierarchical structure together form the service ontology. The 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 proposed approach also supports building service ontology based on operation input and output, respectively. This enriches the view of Web services within a domain, allowing flexible service organization for discovering and composing Web services. These two ontologies can be integrated in a later stage if required.

The remainder of this paper is organized as follows. In Section 2, we give an overview of the proposed framework for building operation-level ontologies for Web services. We lay out and describe the key components in the process. In Section 3, we leverage information retrieval techniques to compare the relevance between operations. We then propose an improved data clustering algorithm to identify service domains and classify operations. In Section 4, we compute the similarity between relevant operations and propose a flexible approach to build up the hierarchical internal structure of service ontologies. In Section 5, we present a comprehensive experimental study to illustrate the effectiveness of the proposed approach. In Section 6, we discuss some representative related work. We conclude our paper and discuss future work in Section 7.

2. ONTOLOGY DEVELOPMENT FRAMEWORK

In this section, we give an overview of the ontology development framework, depicted in Figure 1. The framework consists of several key components, including a *Web service crawler*, *operation extractor*, *operation classifier*, and *ontology constructor*.

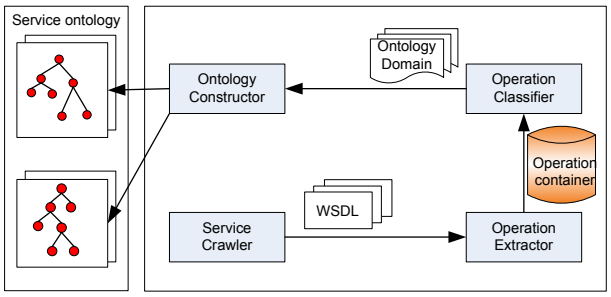

Figure. 1. The Ontology Construction Framework

The Web service crawler is to retrieve WSDL documents by crawling the Web and store them in a service repository. Besides traditional Web search engines, e.g., google and bing, existing Web service search engines, e.g., seekda [1] and programmableWeb [2], can be used as the sources of the service repository. The operation extractor reads WSDL documents from the repository and extracts the description 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

[1] http://webservices.seekda.com/
[2] www.programmableweb.com

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. Therefore, 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 operations' descriptions are stored in an *operation container*.

The operation classifier identifies different service domains by grouping relevance operations together. It first computes 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* [Plebani and Pernici 2009]. 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 3. The operation relevance matrix that stores the relevance between any pair of operations is generated. The operation classifier 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 3.

The ontology constructor computes the similarity between two relevant operations, i.e., the operations in the same domain, and build up internal structure of each ontology. Measuring similarity 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 process is elaborated in Section 4. The ontology constructor then 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 4.

The ontology development framework incorporates 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 [3], 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 [Sahami and Heilman 2006]. Since Wordnet has been demonstrated to be effective to retrieve synonyms for a term, we choose to use Wordnet in our work.

3. SERVICE ONTOLOGY DOMAIN IDENTIFICATION

In this section, we present the process of identifying service domains by grouping related operations together. This process consists of two steps. We first leverage information retrieval techniques to compute the relevance between service operations and generate the relevance matrix. We then propose an improved data clustering approach, addressing the shortcomings of traditional K-Means, to effectively group related operations together. A clustering criterion function is used to optimize the clustering result.

3.1 Modeling Operations

The relevance between two operations is determined based on their descriptions (e.g., name, input message, and output message) and their contexts (Web service names and portType names). In information retrieval techniques, terms are typically used as the basic elements to compare two documents. We leverage this idea for operation comparison. This is due to the observation that the more *representative terms* are shared by two operation descriptions, the higher relevance that the two operations should have. The representative term refers to the one that can represent the functional features of a service operation and it tends to have high appearance frequency among service operations in the same domain. We then use TF/IDF [Baeza-Yates and Ribeiro-Neto 1999] 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. That is, the terms with the same meaning are considered as equivalence.

Let $\mathcal{T}$ be the set of terms extracted from all operation descriptions. It contains k terms with *distinct* meanings. Let $|D|$ be the number of operation descriptions, d be an operation's description. $n'_{j,i}$ be the times that term t_j and all its synonyms appear in ith operation's description. We define the representativeness of a term, i.e., the adjusted TF/IDF, as:

$$\mathbf{r_{j,i}} = tf_{j,i} \times idf_j = \frac{n'_{j,i}}{\sum_k n'_{k,i}} \times \log \frac{|D|}{|d : t_j \in' d|} \tag{1}$$

Here $t_j \in' d$ holds true if term t_j or its synonyms appears in the document.

Based on Equation 1, an operation op_i can be specified as a term vector $< r_{1,i}, r_{2,i}, ..., r_{k,i} >$. The relevance of two relations is computed as the cosine similarity between the two vectors.

$$Rel(op_i, op_j) = \cos(v_i, v_j) = v_i^T v_j / (||v_i|| ||v_j||) \tag{2}$$

3.2 Relevance-based Operation Grouping

From Equation 2, we can generate an operation relevance matrix $\mathcal{M}^R$, where $m^R_{s,t}$=Rel(op_s, op_t). We then apply data clustering techniques to group related operations together. The objective here is to classify operations into several domains (each domain corresponds to a service ontology), so that the operations assigned to each group are more related to each other than the operations assigned to different groups.

K-Means is a widely used centroid-based data clustering algorithm. The basic idea of applying K-Means is firstly randomly choose k operations as initial centroid. k is a predefined number that represents the number of resulting clusters. The algorithm then iteratively clusters operations, computes new centroids (c_i), re-clusters operations based on the new centroids, till centroids do not change. The process of clustering is guided to maximize the cohesion of a cluster, which is

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

defined in Equation 3. K-Means is efficient with time complexity of $\mathcal{O}(KIM)$, where K is the number of clusters, I is the number of iterations, and M is the size of the matrix.

$$cohesion = \sum_{i=1}^{k} \sum_{d \in C_i} \cos(d, c_i) \tag{3}$$

There are two major limitations of K-Means. First, the value of k needs to be predefined. In many cases, it is difficult and even impossible to decide an appropriate value of k. Second, K-Means is susceptible to initialization problem. That is, the result of clustering is heavily affected by the selection of initial centroids. K-Means may result in suboptimal clustering and low cohesion.

To address the above limitations of K-Means, we propose an improved clustering algorithm. We apply a Bisecting K-Means algorithm to solve the initialization problem of K-Means. Instead of predefining k's value, we use silhouette coefficient to determine the termination of the clustering. We observe that the silhouette coefficient increases with the k's value at the first. Once it reaches to the maximum, the corresponding k's value is the most desirable one. During the process of Bisecting iteration, we define a clustering criterion for optimization. We take both maximizing *intra-domain relevance* and minimizing *inter-domain irrelevance* into account. The Intra-Domain Relevance ($\mathcal{R}$) is the sum of the average pairwise relevances between the operations assigned to each domain. It is defined as:

$$\mathcal{R} = \sum_{r=1}^{k} (\frac{1}{n_r} \sum_{d_i, d_j \in \mathcal{C}_r} \cos(d_i, d_j)) \tag{4}$$

The Inter-Domain Relevance ($\mathcal{T}$) is the sum of the relevances between the centroid operation of each cluster and the centroid operation of the entire operation set. It is defined as:

$$\mathcal{T} = \sum_{r=1}^{k} n_r \cos(c_r, c) \tag{5}$$

Therefore, our ultimate goal of clustering operation is to maximize the ratio between $\mathcal{R}$ and $\mathcal{T}$. That is, the cluttering criterion for optimization is defined as:

$$\mathcal{O} = \frac{\sum_{r=1}^{k} (\frac{1}{n_r} \sum_{d_i, d_j \in \mathcal{C}_r} \cos(d_i, d_j))}{\sum_{r=1}^{k} n_r \cos(c_r, c)} \tag{6}$$

Algorithm 1 An Improved Algorithm for Operation Clustering

1: **Function Operation_Clustering**($\mathcal{R}$, $\mathcal{D}$) {$\mathcal{R}$ is the operation-term matrix, $\mathcal{D}$ is the operation relevance matrix}
2: max_Silhouette_Coefficient=0; objective_k=-1; Clustering_Result={ϕ};
3: k=1; Clusters=C_0, C_0 contains all the operations
4: **while** (1)
5: $k = k + 1$;
6: $\mathcal{C}$=max_Cluster(Clusters) {choose the largest cluster from the cluster list}
7: **for** j=1 to ITER do {ITER is the number of trials}
8: Use K-means to split C to two subclusters
9: **endfor**
10: select the two clusters from the bisect with the highest $\mathcal{O}$
11: Add the two clusters to Clusters
12: c=Compute_Silhouette_Coefficient (C);
13: **if** (c $\geq$ max_Silhouette_Coefficient)
14: max_Silhouette_Coefficient=c; objective_k=k; Clustering_Result=Clusters;
15: **endif**
16: **if** (c $\leq$ max_Silhouette_Coefficient)
17: break;
18: return objective_k, Clustering_Result;
19: **Function Compute_Silhouette_Coefficient(Cluster c)**
20: **for all i**=1 to n {n is the number of operations}
21: a_i=Average distance to all other operations in its cluster
22: b_i=minimum (average distance to all the operations in a cluster that the ith operation does not belong)
23: s_i=$|(b_i - a_i)|/\max(a_i, b_i)$
24: **endfor**
25: result=average of s_i; return result

The process of improved operation clustering is described in Algorithm 1. We use Bisecting K-Means algorithm to classify operations into k clusters, where k starts at 1. The bisecting process starts with one cluster that contains all operations. It then chooses the largest cluster and splits it into two sub clusters applying K-Means for multiple times (ITER defines the time of iterations in Line 17). The result is optimized by using the objective function defined in Equation 6 (in Line 20). The process is repeated, which causes the increasement of k, until *Silhouette Coefficient* achieves the maximum value. We will choose k with the highest Silhouette_Coefficient value as the number of operation clusters.

4. SIMILARITY-BASED ONTOLOGY CONSTRUCTION

In this section, we describe the process of similarity-based ontology construction. This process is to build up hierarchical structure of a service ontology based on the functional similarity between operations. Two operations are considered to be functional similar if given the same input, they will output the similar output. The degree of functional similarity can be measured by comparing three dimensions: operations' name, input, and output, respectively. The placement of a term, such as whether it appears in name, input, or output, will directly affect the result of measurement. This makes it different from measuring operations relevance where the placement of terms is irrelevance.

We exploit a similar strategy as developed in [Plebani and Pernici 2009] to measure similarity between two service operations in each dimension. Based on operation similarity, we propose two strategies to build up service hierarchical structure. In the first strategy, we assign different weights to the similarity in different dimensions and then aggregate them as the overall one. We then apply hierarchical data clustering algorithm. In the second strategy, we build up a hierarchical clustering of operations for each dimension. These clusterings can be ensembled

together in a later stage to generate an overall clustering. The second strategy provides more flexibility of generating hierarchy of service operations, organizing operations based on their name, input, and output respectively.

4.1 Computing Operation Similarity

The similarity of operations are measured by comparing their names, input, and output, respectively. When comparing two operations' names, we need to first tokenize the names to a set of terms. This is due to the observation that an operation typically has a composite name, such as get_Purchase_Order or carRentalReservation. 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 $op_j.name$,

$$simName(op_i.name, op_j.name) = simTermSet(T_{op_i}, T_{op_j}). \tag{7}$$

We model the two sets of terms as two sets of nodes in a bipartite graph and compute the maximum weight matching as their similarity. 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 [Cilibrasi and Vitanyi 2007]. We first compute the normalized compression distance (NCD) between two terms. We then get the similarity as:

$$simTerm(t_1, t_2) = 1 - NCD(t_1, t_2) \tag{8}$$

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., $\{< t_{i1}, t_{j1} >, < t_{i2}, t_{j2} >\}$.

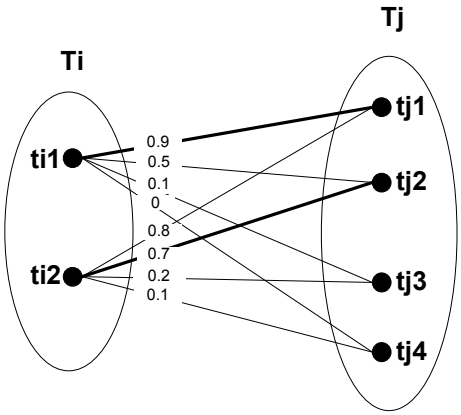

Figure. 2. The bipartite graph model of two term sets

The weight of a matching is:

$$W_M = \sum_{i=1}^{i=|M|} simTerm(e_i.from, e_i.to) \tag{9}$$

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

$$simTermSet(T_{op_i}, T_{op_j}) = \frac{\max_{1 \leq k \leq n}(W_{M_k})}{\min(|T_{op_i}|, |T_{op_j}|)}, \tag{10}$$

We use Equation 7 and 8 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:

$$\begin{aligned} simMsg(m_i, m_j) = \\ & w_1 \times simName(m_i.name, m_j.name) + \\ & w_2 \times simMsg(m_i.Parts, m_j.Parts) \\ & \text{where } w_1 + w_2 = 1. \end{aligned} \tag{11}$$

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:

$$\begin{aligned} simPart(ap_1, ap_2) = \\ & w_1 \times simName(ap_1.name, ap_2.name) + \\ & w_2 \times simType(ap_1.type, ap_j.type) \\ & \text{where } w_1 + w_2 = 1. \end{aligned} \tag{12}$$

4.2 Constructing Internal Structure of Service Ontology

The process of constructing internal structure of service ontology is to define a view of the service functionality within a domain at different levels of abstraction. Each level of abstraction can be mapped to a set of operations which provide the corresponding functionality. This makes hierarchical clustering [Sibson 1973] a perfect solution for this process. In addition, different from other clustering algorithms, such as K-Means where "centroids" need to be computed for clustering, several hierarchical clustering algorithms, such as SLINK algorithm, cluster 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".

Our first strategy is to assign weights to the similarity on the three dimensions and then aggregate them as the overall similarity. That is,

$$\begin{aligned} simOP(op_i, op_j) = \\ & w_1 \times simName(op_i.name, op_j.name) + \\ & w_2 \times simMsg(op_i.in, op_j.in) + \\ & w_3 \times simMsg(op_i.out, op_j.out) \\ & \text{where } w_1 + w_2 + w_3 = 1. \end{aligned} \tag{13}$$

Through Equation 13, we can compute the similarity between any pair of operations, which results in a similarity matrix. We then apply SLINK to build the hierarchy based on the similarity matrix.

The major issue of this approach is that the resulted service ontology heavily relies on the weight assignments. Inappropriate assignment of weights will lead to a poor clustering result. Moreover, the clustering process is based on an aggregate value of operation similarity. In case that the weight assignment changes, which may be caused by dynamic ontology construction

requirement, the operation similarity needs to be recomputed and the clustering process needs to start over. This will get worse if the intermediate result of similarity measurement is not stored, which is quite typical.

To address this issue, for each dimension (i.e., name, input message, output message), we apply SLINK to generate single-faucet operation clusterings. The overall service ontology can be achieved by ensembling these three clusterings together without recomputing the similarity. This approach has two major benefits. First, the single-faucet operation clusterings provide more insights into the service organization. That is, services are organized by their names, input, and output, respectively. Second, once the weight assignment changes, the service ontology can be easily regenerated by leveraging the three existing clusterings, instead of going through the whole process of ontology construction again. This improves the overall performance.

We introduce the dendrogram descriptors that will be used in our approach, including *Cophenetic Difference* (CD), *Maximum Edge Distance* (MED), *Partition Membership Divergence* (PMD), *Cluster Membership Divergence* (CMD), and *Sub-dendrogram Membership Divergence* (SMD) [Mirzaei et al. 2008]. CD refers to the lowest merge distance of internal nodes in the dendrogram where two specified leaves are joined together. It can be recorded as the number of the iteration when two operations are clustered together. MED refers to the depth of nodes. All leaf nodes have depth as 0. Any internal nodes has depth as the maximum depth of its children plus one, i.e., Depth(N_1,N_2)=max(Depth(N_1), Depth(N_2))+1. PMD refers to the number of partitions of hierarchy in which two specific leave nodes are not in the same cluster. CMD refers to the size of the smallest cluster in the hierarchy in which contains two specified leaves. SMD refers to the number of sub-dendrograms in which two specified leaves are not included together.

The above dendrogram descriptors can be used to generate similarity based description metrics, which can be used as the matrix representation of dendrogram. If D is a dissimilarity relation of n operations, we first normalize the entries in D and then generate the similarity matrix S as :

$$S = [1]_{(n\times n)} - D' \tag{14}$$

Given three dendrograms that correspond to three clusterings, we apply alpha-cut to obtain the final hierarchical clustering. The idea is to combine the three representation matrices so that the consensus matrix has an associated dendrogram. For this purpose, we compute the min-transitive closure of similarity matrix to form the consensus matrix which is transitive. This is to make sure that the dendrogram can be generated from the consensus matrix. This process is described in Algorithm 2.

We first generate the similarity matrix for each dendrograms (Line 2-3). We then aggregate the similarity matrices (i.e., $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$) to a final one. This process is done by repeatedly performing matrix composition until a min-transitive combination of the dendrograms is achieved (Line 4-10). In Line 8, $(S^* \circ D^i)$ is the composition of two fuzzy relations. It is defined as:

$$\beta_{S^*\circ D^i} = \max_{z\in X}\{\min\{\beta_{S^*}(x,z), \beta_{D^i}(y,z)\}\} \tag{15}$$

We can then generate the final clustering from S by performing the α-cut of S (Line 11).

Algorithm 2 Hierarchical Clustering Ensemble

Function Hierarchical_Clustering_Ensemble($\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$),
for all i from 1 to 3
  D^i=Similarity_based_Metrix($\mathcal{D}_i$);
S^* is the Identify matrix,
while (1)
  $S^0 = S^*$
  for all i from 1 to 3
    $S^* = S^* \cup (S^* \circ D^i)$
  if S^0=S^*
    S=S^*;break;
$D^F = f^{-1}S$ {This is done by drawing the α-cut of S}

5. 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 [4]. 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.

5.1 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. We implement our improved operation clustering algorithm based on Bisecting K-means and set the ITER as 8. We compute the silhouette coefficient for k from 2 to 10. As can be seen from Figure 3, we have the maximum silhouette coefficient when k equals to 5. This conforms to the structure of the data set. This shows that our algorithm can detect the best k value.

We then evaluate the accuracy of the clustering result. The precision and recall are calculated as follows:

$$\text{precision} = \frac{\text{the number of correctly clustered operations}}{\text{the total number of operations clustered into the ontology}} \tag{16}$$

$$\text{recall} = \frac{\text{the number of correctly clustered operations}}{\text{the total number of operations from the given domain}} \tag{17}$$

We compare our algorithm with K-means, which is a widely used data clustering algorithm. As can be seen, As can be seen from Table I, better precision and recall are achieved by improved algorithm than K-means. For three ontologies: communication, medical, and education, we achieve perfect result. Only one operation from food ontology is mis-clustered to travel ontology. However, the K-means algorithm mis-clusters 8 travel operations into the medical ontology.

5.2 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.

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

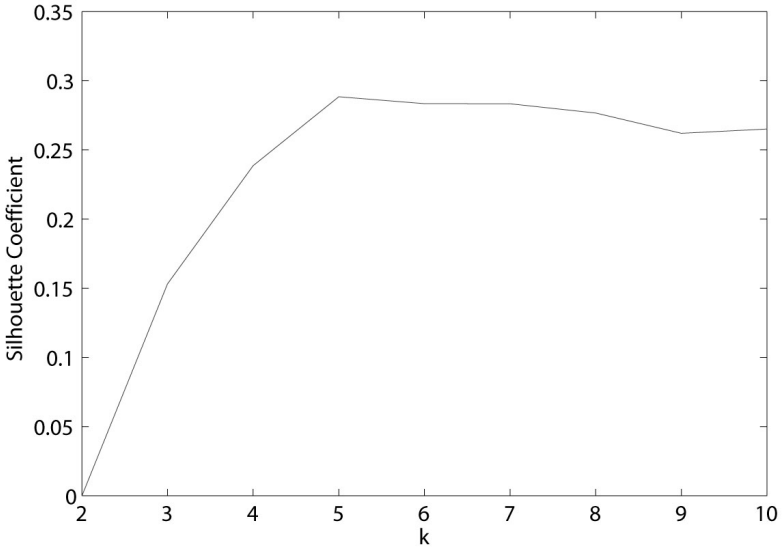

Figure. 3. The relationship between k and silhouette coefficient

Table I. Result of Relevance-based Clustering

Ontology name	**Improve algorithm Precision**	**Improve algorithm Recall**	**K-means Precision**	**K-means Recall**
Communication ontology	100%	100%	100%	100%
Food ontology	100%	94%	100%	100%
Medical ontology	100%	100%	70.4%	100%
Travel ontology	95%	100%	100%	60%
Education ontology	100%	100%	100%	100%

Figure 4 shows the travel ontology structure obtained from hierarchical clustering of operations in the travel ontology. For the sake of space, we only present the clustering result of the first strategy, i.e., aggregating the similarity on three dimensions. 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 4, 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

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)]

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, 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 achieved 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.

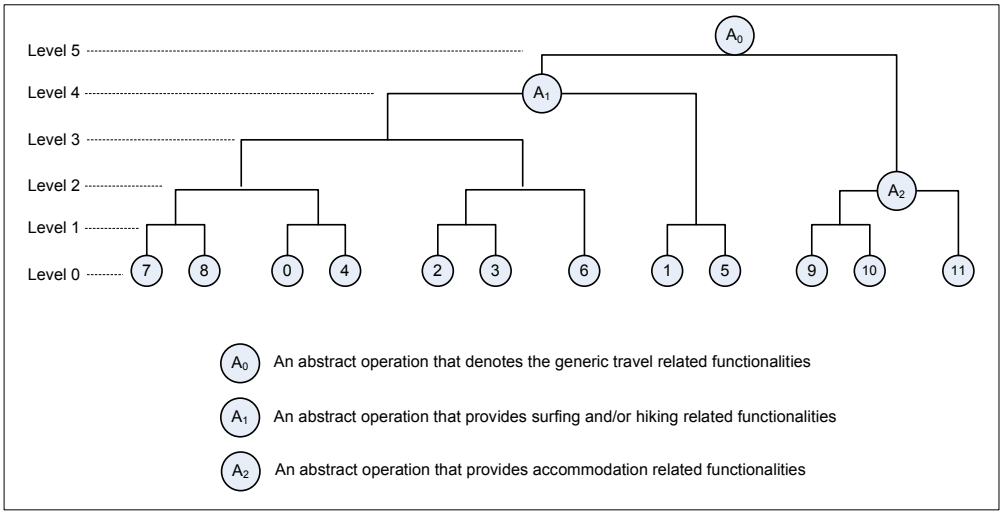

Figure. 4. The Hierarchical Travel Ontology Structure

6. 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.

6.1 Ontology Development

The approach proposed in [Segev and Sheng 2010] 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 [Wu et al. 2005], 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 [Segev and Sheng 2010] and [Wu et al. 2005] mainly focus on domain ontologies learning. Domain ontologies capture the data-level concepts and their relationships. They can be used to semantically annotate service description. However, they do not provide an integrated view on service functionality and inter-service relationships. Our work, on the other hand, focuses on developing service ontologies, where services, more specifically, service operations, are treated as the first-class objects. Service ontologies define a functionality-based service organization, which allows a structured way of discovering and composing Web services.

6.2 Semantic Extraction from WSDL

In [Yu and Rege 2010], 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 [Elgazzar et al. 2010] 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 [Yu and Rege 2010] and [Elgazzar et al. 2010] 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 [Elgazzar et al. 2010] 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 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 [Plebani and Pernici 2009], 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 [Liu et al. 2010], an approach for measuring similarity between two WSDL documents is proposed aiming to improve the work in [Plebani and Pernici 2009]. 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 [Plebani and Pernici 2009]. The first phase similarity follows the same process proposed in [Plebani and Pernici 2009]. In the second phase, the unmatched terms and their weights are considered to compute the similarity.

The work proposed in [Plebani and Pernici 2009] and [Liu et al. 2010] 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.

7. CONCLUSION

We present an integrated framework that automatically develops ontologies for Web services. The ontology construction is based on the computation of relevance and similarity of service operations. We model service operations as term vectors and propose an improved data clustering algorithm to efficiently and effectively 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 hierarchical data clustering algorithm to build up the internal structure of the ontology.

There are two directions for the future work. First, we plan to apply our approach to other service description formats, such as OWL-S, where richer information (e.g., semantic markups, preconditions, and effects) can be used to improve the quality of resulted service ontologies. Second, service ontologies need to be evolved due to the appearance of new services, i.e., the structures of service ontologies need to be changed to incorporate the new services. It is not practical to construct service ontologies from scratch every time when a new service comes in. We plan to work on an efficient and effective approach for ontology evolvement.

REFERENCES

Akram, S., Bouguettaya, A., Liu, X., Haller, A., and Rosenberg, F. 2010. A change management framework for service oriented enterprises. *IJNGC 1,* 1.

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

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

Coalition, T. O. S. 2004. Owl-s: Semantic markup for web services. http://www.daml.org/services/owl-s/1.1B/owl-s/owl-s.html.

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

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

Liu, X. and Bouguettaya, A. 2007. Managing top-down changes in service-oriented enterprises. 1072–1079.

Liu, X. and Liu, H. 2011. Bootstrapping operation-level web service ontology: A bottom-up approach.

Medjahed, B., Bouguettaya, A., and Elmagarmid, A. K. 2003. Composing web services on the semantic web. *The VLDB Journal 12*, 333–351.

Mirzaei, A., Rahmati, M., and Ahmadi, M. 2008. A new method for hierarchical clustering combination. *Intell. Data Anal. 12,* 6, 549–571.

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

Rezgui, A., Ouzzani, M., Bouguettaya, A., and Medjahed, B. 2002. Preserving privacy in web services. 56–62.

Sahami, M. and Heilman, T. D. 2006. 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.

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

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

W3C. 2001. Web services description language (wsdl). In *http://www.w3.org/TR/wsdl*.

W3C. 2005. Web service semantics - wsdl-s. In *http://www.w3.org/Submission/WSDL-S/*.

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

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

Yu, Q. and Bouguettaya, A. 2008. Framework for web service query algebra and optimization. *ACM Trans. Web 2,* 1, 1–35.

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

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

Dr. Hua Liu is a senior research scientist in Xerox Research Center at Webster. Her research interests include service-oriented computing, cloud computing, and autonomic computing. In these research areas, Dr. Liu has 3 patents, 12 patent applications, and 20+ publications with near 200 citations. Dr. Liu was on the editorial board of IJSIA (International Journal of Security and Its Applications) and the Open Cybernetics and Systemics Journal, and served on the programming committee of DATA, ICSOFT, DEXA, ARES, SMC, ISA, DASC and ICDCS. Dr. Liu obtained a Ph.D. in Computer Engineering from Rutgers University in 2005. Dr. Lius latest information can be found at http://www.linkedin.com/in/hualiu.

Xumin Liu is an Assistant Professor in the department of computer science at Rochester Institute of Technology. She received her PhD in computer science from Virginia Tech. Her research interests lie in the general field of data management and service computing with special focuses on change management, semantic Web services, and service composition. Dr. Liu's research has been published in various international journals and conferences, such as the International Journal on Very Large Data Bases (VLDB journal), IEEE Transactions on Services Computing (TSC), International Journal of Web Services Research (IJWSR), and IEEE International Conference on Web Services (ICWS). Dr. Liu frequently serves as a program committee member or a reviewer for various international conferences. She is also a reviewer for various journals, including IEEE Transactions on Knowledge and Data Engineering (TKDE) and IEEE Transactions on Services Computing (TSC).