Discovering Homogeneous Web Service Community in the User-Centric Web Environment

Xuanzhe Liu, Gang Huang, Mei Hong

Discovering Homogeneous Web Service Community in the User-Centric Web Environment

Xuanzhe Liu, Gang Huang, *Member*, *IEEE*, and Hong Mei, *Senior Member*, *IEEE*

Abstract—The Web has undergone a tremendous change toward a highly user-centric environment. Millions of users can participate and collaborate for their own interests and benefits. Services Computing paradigm together with the proliferation of Web services have created great potential opportunities for the users, also known as service consumers, to produce value-added services by means of service discovery and composition. In this paper, we propose an efficient approach to facilitating the service consumer on discovering Web services. First, we analyze the service discovery requirements from the service consumer's perspective and outline a conceptual model of homogeneous Web service communities. The homogeneous service community contains two types of discovery: the search of similar operations and that of composible operations. Second, we describe a similarity measurement model for Web services by leveraging the metadata from WSDL, and design a graph-based algorithm to support both of the two discovery types. Finally, adopting the popular atom feeds, we design a prototype to facilitate the consumers to discover while subscribing Web services in an easy-of-use manner. With the experimental evaluation and prototype demonstration, our approach not only alleviates the consumers from time-consuming discovery tasks but also lowers their entry barrier in the user-centric Web environment.

Index Terms—Web services discovery, service metadata, service community.

1 Introduction

The Web keeps rapidly growing in recent years. Current Web has been a "user-centric" environment where millions of users can participate and collaborate for their own interests and benefits [1]. The services computing paradigm together with the proliferation of Web services make the Internet as a huge resource library, and millions of users can participate and create more value-added services by means of service discovery and composition. Existing SOA technologies, including Universal Description, Discovery, and Integration (UDDI) and service composition languages (such as BPEL4WS), have fostered agile integration by simplifying integration at the communication, data, and business logic layers. Furthermore, by leveraging efforts in semantic Web services, service composition frameworks made a forward step on enabling automated support for service description matching.

Service discovery is a significant activity in Services Computing paradigm. Efficient discovery plays a crucial role in conducting further service composition. With the ever increasing number of services over Internet, more and more service consumers (including nonexpert users, Small and Medium Enterprise, and transient business partners of specific opportunities/interests) can participate in the composition activity[1] [1]. Meanwhile, a key problem also matters locating the desired services efficiently. Although existing discovery techniques have produced promising results that are certainly useful, they may not well aligned with the needs of Internet-scale environment. First, searching Web services via some public UDDI registries [35], [36], [37], [38] is mainly based on the keywords involved in query and matches them with the Web service descriptions. As the keywords are not able to capture the underlying semantics, they may miss some results and return a lot of irrelevant ones as well. Second, the users would like to specify their requests more precisely rather than just keywords. Actually, searching Web services is searching for the operations offering some functionalities, and current discovery usually explores details of the service operations. The service consumers have to browse each returned results in detail and check if they meet their requirements or not. Nevertheless, investigating a single operation usually needs several steps. Hence, the more service providers emerge, the heavier burden it brings to the consumers. Third, the Web services are developed and maintained by their providers. For some reasons, such as the market competitions and cost control policies, the providers may update or

- *X. Liu is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Room 1616E, Science Building 1, Beijing 100871, China, and the Institute of Software, School of Electronics Engineering and Computer Science (EECS), Peking University. E-mail: liuxzh@sei.pku.edu.cn.*
- *G. Huang is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Room 1809S, Science Building 1, Beijing 100871, China, and the Institute of Software, School of Electronics Engineering and Computer Science (EECS), Peking University. E-mail: huanggang@sei.pku.edu.cn.*
- *H. Mei is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Room 1620E, Science Building 1, Beijing 100871, China, and the Institute of Software, School of Electronics Engineering and Computer Science (EECS), Peking University. E-mail: meih@pku.edu.cn.*

Manuscript received 11 Dec. 2008; revised 29 Mar. 2009; accepted 24 Apr. 2009; published online 30 Apr. 2009.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2008-12-0109.
Digital Object Identifier no. 10.1109/TSC.2009.11.

1. In this paper, we use the terms "service consumer" and "service user" exchangeably.

remove their Web services at any time. Once the Web services are modified or even no longer available, the service consumers have to repeat the discovery process to find new appropriate services. To best of our knowledge, current discovery approaches cannot deal with the ever-changing Web services.

Due to the reasons above, we argue that current service discovery significantly prevents its ubiquitous adoption among Internet users. Much simpler and more efficient service discovery is required in the user-centric and demand-driven Internet environment in order to lower the entry barrier for the service consumers.

To meet these requirements, we first go back to the success of the Web community and the reluctance in taking up the Web service idea, it is inevitable for us to review the fundamental basis of SOA. As everyone can publish a Web page with a valid URL, there is really no need to "register" the Web page. Various search engines automatically trawl a set of Web pages and classify them into groups. Moreover, the search engine can even retrieve the access path according to the hyperlinks, which looks like the "composition" of the Web pages. Such search manner makes Web search engine widely adopted by the Internet users. Similarly, in the area of Web service discovery, once the consumers drill down all the way and find the Web service inappropriate for some reason, they may prefer being able to find a set of similar operations that takes similar inputs/outputs to the ones just considered, instead of laboriously browsing them one after another [2]. What's more, they may intend to find operations that can be composed with the current ones being browsed. Therefore, it seems to be reasonable to support search for similar Web service that can do the job of clustering, classification, match-making, and composition. Such manner will be more free and efficient for the consumers to find their desired services.

Another promising hint comes from the innovations of Web 2.0 wave. In our investigation, the most prevailing Web 2.0 communication mechanism is not the complex "centralized registry," but lightweight manner such as the RSS [31] or Atom [32]. With the feeds, the users are able to organize several Web pages such as news or Weblogs to a specific topic/interest and subscribe them. Once the resources are changed or updated, the RSS/Atom will notify their subscribers. Similarly, if Web services can be organized into RSS/Atom feeds and subscribed by the service consumers, it will release the work of locating the ever-changing Web services. On the other hand, since most current Web browsers (such as Microsoft Internet Explorer, Firefox, and Safari) all support RSS/Atom, it is then feasible to provide a universal and convenient channel for the service consumers, which allows them to easily locate desired Web services and further participate in the service composition.

Our work aims to provide simpler and more efficient Web service discovery, which may align the requirements of the user-centric Web environment [4], [5], [6]. In this paper, we propose an approach to efficient and easy-of-use Web service discovery. Beyond the keyword search, our approach tries to assist the service consumers to find the similar service operations as well as the potentially composible ones, against the given requests. As both of these two types represent the services offering similar functionalities, we term them as *homogeneous Web service communities* in this paper. The main contributions of this paper are the following:

- We investigate the similarity measurement to retrieve a set of Web services into homogeneous service communities. We consider the basic metadata (inputs, outputs, and operations within a message) that resides in the WSDL file, and propose mining algorithms to retrieve their underlying semantics. The key idea is to measure the co-occurrence of terms and cluster the terms into a set of concepts, and leverage these concepts to determine the similarity of inputs/outputs and operations.
- We design a graph-based search model to find both the similar operations and the composible ones, according to the user's requests. In our model, every operation is represented by a vertex in the graph and the potential composition opportunities are represented by the directed edges. Then, the discovery of similar operations is transformed to the traversal of vertexes, and that of composible operations is transformed to find the corresponding paths. To evaluate the approach, we set up a series of experiments on the data set of over 500 real-world Web services. The results prove that approach has very high recall and precision.
- We present the design of a prototype to evaluate the usability of our approach. Within the local Web browser, the consumers can intuitively do search for the homogeneous service communities. We particularly discuss the implementation details of how the Web services are mapped to the Atom feeds. Hence, the discovered Web services can be organized and subscribed like regular Web pages, and the consumers are able to track their changes. It not only facilitates the consumers to deal with the ever-changing Web services, but also reduces the complexity of participating and collaborating the discovery in the user-centric Web environment.

The remainder of this paper is organized as follows: Section 2 introduces the overview of our approach, by identifying the fundamental rationales and challenges. Section 3 introduces the similarity measurement algorithm, which mines the metadata from WSDL and clusters them to concepts. The concepts are leveraged to express the underlying semantics to determine the similarity. Section 4 introduces the graph-based algorithm to find the homogeneous service communities. We describe the construction issues of the ***Service Aggregation Graph***, and how to search both single similar operations and composible operations. Section 5 describes some experiments and evaluates the efficiency of our approach. Section 6 presents the prototype architecture, particularly focuses on the implementation details of how map Web services into Atom feeds. Section 7 discusses the related work, and Section 8 ends the paper with the conclusion and future work.

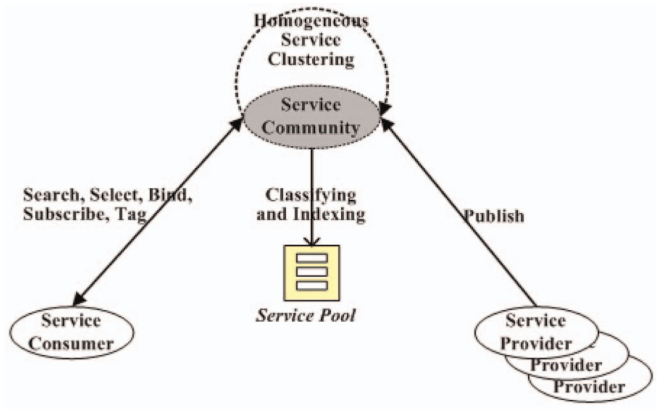


Fig. 1. Conceptual overview of ServicePool-based discovery.

2 Approach and Challenges

In this section, we are going to present the conceptual overview of our approach as well as the corresponding rationales and technical challenges.

2.1 Approach Overview

To begin with, let's consider the typical scenario when searching for Web services [2]. Users commit a search for Web services by typing in keywords relevant to the search goal. They then start inspecting some of the returned Web services. Normally, the result is rather complex for it includes the description of Web services, the operations, input/output parameters, and their data types, so the users have to drill down in several time-consuming steps. They decide which Web services to explore in detail, and then, consider which specific operation in the service to look at, for the operation is exactly the information needed to build their own applications. Given a particular operation, they will look at each of the inputs and outputs specified in the "message" element. Once the users may find that the Web service is inappropriate for some reason, instead of repeating the same process for each of other potentially relevant services, they may want to get a list of operations that can process the same/similar inputs/outputs, or moreover, a sequence of operations to compose with given requests.

Based on such observation, we show the conceptual model of the ideal search scenarios, as shown in Fig. 1. The service providers develop and publish Web services as usual, e.g., to some registries or via valid URL. With some clustering algorithm, we employ a mediator that is able to aggregate the similar Web services into a so-called "ServicePool" [4], [5]. This ServicePool not only maintains the operations with similar inputs/outputs, but also the "hyperlink" between the operations. Then, once the service consumer searches the ServicePool, they do not have to take the time-consuming steps above. Furthermore, with current Web 2.0 technologies, they can even subscribe, tag, and blog the results, and even organize the results in their own manner.

2.2 Rationales and Challenges

The conceptual model promises a simple and direct discovery manner. When searching the homogeneous Web services, we can identify the following two types to find the similar Web services.

First, the users would like to get the set of "single" Web services with similar functionalities. To the best of our knowledge, the functionalities offered by a Web service are usually reflected by its operation. For example, the weather report services may provide operations such as "*GetWeatherByZipCode*" or "*GetTemperatur*e." Therefore, the problem can be viewed as "searching for similar operations." Intuitively, the operations are similar if they have similar inputs, generate similar outputs, and the relationships between the inputs and outputs are also similar [2].

Second, if no single service operation is qualified for the request, the users may also want to retrieve a sequence of operations that can be composed together. It means that the outputs generated by one service can be accepted as the inputs of another service. For example, suppose two Web services S_1 and S_2. S_1 is an "Address Querying" service which can output the city name according to the given zip code (e.g., the output is "*ZipCodeToCity*"), and S_2 is a "Weather Forecasting" service which can return the weather forecast by the given city name (e.g., the input is "*City,*" "*State*"). S_1 and S_2 can be composed together in case that the city name is unknown. Such process can iteratively proceed to construct "operation hyperlink" until the desired result is fulfilled.

Both of the two discovery types mentioned above are essentially related to the similarity measurement of Web service operations, inputs, and outputs. Like traditional clustering approaches for Web pages or topics, we are going to employ similarity measurement to retrieve the homogeneous Web services. Then, some technical challenges need to be solved.

As is known to all, semantics means crucially to determining the similarity. But in current WSDL specification, neither the textual descriptions of Web services and their operations nor the names of input/output parameters completely convey the underlying semantics of the service operation. Therefore, searching for similar Web services is much more challenging. To efficiently match the inputs/outputs of Web service operations, it is then important to get their underlying semantics. By investigating the metadata from the WSDL structure, our approach tries to combine multiple sources of evidences to determine the similarity between Web services. In Section 3, we describe a mining algorithm that clusters metadata (including input/output names) from a collection of Web services into some semantically meaningful concepts. By comparing the concepts, they belong to, and considering the similarity of the descriptions of the operations and the entire Web services, we can have a good similarity measurement.

Besides the similarity measurement, we may need a search model. Within the ServicePool, we hold two types of search: single similar operations and composible operation sequences. Thus, the search model is expected to be able to process both of the two types, and promise high efficiency. In Section 4, we merge these two types by employing an algorithm based on a Directed Graph. We make each operation as a vertex, maintain the composition opportunities as directed edges, and assign the weight of the edge with similarity matching score between inputs and outputs. Then,

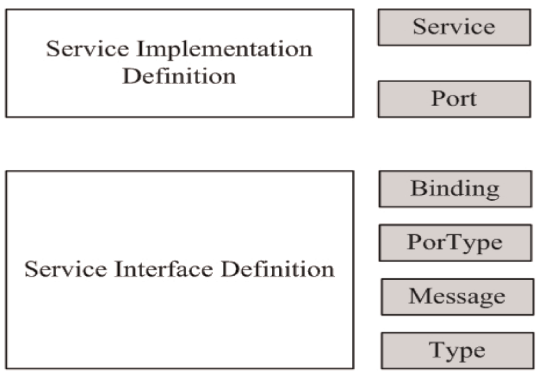

Service Implementation Definition
Service
Port
Service Interface Definition
Binding
PorType
Message
Type

Fig. 2. WSDL Structure.

the search for the single similar operations is transformed to the traversal of all vertexes, and the search of composible operations is transformed to find the corresponding paths in the graph. We analyze the complexity of our algorithm and prove that it can work with high efficiency.

Finally, as ServicePool is supposed to be an aggregator of a group of homogeneous services, such organization style is similar to the news group that can be subscribed with RSS or Atom. Moreover, with the RSS/Atom, the consumers will be notified once their subscribed Web services are changed. Therefore, we need to bind the WSDL to the RSS/Atom feeds while considering the semantic consistency. Particularly, we find that the current Atom specification has already defined some useful elements for Web service discovery and subscription. For example, each Atom entry has a unique ID, while a Web service has also one to identify itself while regardless the version, location, and invocation information. So, the entry ID is useful when subscribing the "service feed." Another example is "atomlink," which is used for the entry and feed. It explicitly defines a mechanism which can flexibly attach some customized metadata into RSS. Thus, when the entry is a WSDL file or OWL-S represented by RDF, we can then import several "Atomlink" to describe the MIME types. The service consumer can retrieve these data in a standard manner via HTTP. The design details will be illustrated in Section 6.

3 Web Service Metadata Clustering

In this Section, we introduce how to measure the similarity between Web services. Our approach combines multiple sources of evidence to determine the similarity. First, we go to investigate the metadata from the WSDL files.

3.1 Service Metadata Model

A Web service is described in an XML-based document, called WSDL. The WSDL specifies the service implementation definition and service interface definition, as shown in Fig. 2. The service implementation definition describes how a service interface is implemented by a given service provider, and the service interface definition contains the business category information and interface specifications that are registered as UDDI tModels. Generally, we can define a Web service as follows.

Definition 1. *A Web service $WS = < N, M, D, O >$, where N is the name that specifies a Web service, D is the set of data types, M is the set of messages exchanged by the operation invocation, O is the set of operations provided by the Web service, and in which:*

- *an operation op is a vector $op = < n_{op}, input, output >$, that is, $\forall op \in O$ is described by a name and a text description n_{op} in WSDL defined by the portType element and associated with at least one input message and one output message. Note there is a set of operations in a WSDL.*
- *Input/Output: $\forall input \in M (\forall output \in M)$ is a message m. Each input and output contain a set of parameters for an operation defined by the message element and the type element used in the message (for representing the complex data types).*
- *a message $m \in M$ has optionally $\mathrm{i}(\mathrm{i} \geq 1)$ parts, represented as $m = \{d_1, d_2, \ldots, d_k\}$, where $d_j \in D$, $1 \leq \mathrm{j} \leq \mathrm{k}$.*

From the definition, regardless of the invocation information in WSDL that is useless for similarity matching, such as the binding and the port, we can identify three types of metadata from WSDL. First, we note the plain textual descriptions, which describe the general kind of service that is offered, for example, service related to "weather forecasting" or "travel agency." Second, we note the domain of the operation that captures the purposed functionality, such as "GetWeather ByZipCode," "SearchBook," or "QueryAirplaneTimetable." Finally, we find the data type deriving from the input/output. The data types do not relate to the low-level encoding issues such as integer or string, but to the semantic meanings such as "weather," "zipcode," etc.

3.2 Estimating the Parameters

In terms of similarity measurement, the service descriptions can be easily determined by the traditional Term Frequency and Inverse Document Frequency (TF/IDF) methods. However, the similarity of operations and inputs/outputs cannot be determined. On one hand, the parameter naming is mostly dependent on the service provider/developer's personal habit. Hence, parameters tend to be highly varied given the use of synonyms, hypernyms, and different naming rules. On the other hand, inputs/outputs typically have very few parameters, and the associated WSDL files rarely provide rich description.

As proposed existing work in [2], we try to explore the underlying semantics of the inputs/outputs in addition to their textual descriptions. First, an intuitive heuristic is that the parameter names, which are specified in the inputs/outputs and operations, are often combined as a sequence of several *terms*. Take the parameter *"GetWeatherByZipcode,"* for example, the terms are specified by their first letter capitalized *{Get, Weather, By, Zipcode}*. We cluster these terms into several concepts. In our opinion, considering the terms with the concepts they belong to may significantly improve the similarity measurement. For example, given the two outputs *{weather}* and *{temperature, humidity}*, they cannot be considered to be similar just by checking with their names. But these terms are all related to the concept of "weather," they should be similar data types.

3.3 Clustering Concepts from Inputs and Outputs

When clustering metadata residing in the input/output data types into several meaningful semantic concepts, we intuitively consider the words co-occurrence. *A common sense heuristic is that the words tend to express the same semantic concept if they often occur together* [17]. In other words, similar data types tend to be named by similar names, and/or belong to messages and operations that are similarly named. Therefore, we can then exploit the conditional probabilities of occurrence.

Let $I = \{t_1, t_2, \ldots, t_m\}$ be a set of terms. These terms come from term bags to which similarity measurement is applied. Let D be a set of candidate Web service input/output descriptions available from WSDL files. To reflect the co-occurrence, we then introduce association rules. Generally speaking, an association rule is an implication of the form $t_i \rightarrow t_j$, where both t_i and $t_j \in I$. The rule $t_i \rightarrow t_j$ holds in the description set D with support s and confidence c.

Definition 2. *The support* $s = \Pr(t_i) = N(t_i)/N(D)$ *is the probability that* t_i *occurs in an input/output, where N(D) is the total number of inputs or outputs of D and N(*t_i*) is the number of inputs or outputs that contain ti.*

Definition 3. *The confidence* $c = \Pr(t_i, t_j) = N(t_i, t_j)/\ N(t_i)$ *is the probability* t_j *that occurs in an input/output, given* t_i *is already known to occur in it, where N(*t_i*) is the number of inputs or outputs that contain* t_i *and* $\mathrm{N}(t_i, t_j)$ *is the number of inputs or outputs that contain both* t_i *and* t_j*.*

All association rules can be found by the well-known A-Priori algorithm [23]. Obtaining the association rules for the term set, we then try to cluster the concept set. We exploit the co-occurrence of words in word bags to cluster them into meaningful concepts. We begin with each term forming its own cluster and gradually merge similar clusters.

Given a threshold t, we use the agglomeration algorithm to generate the terms set $I = \{t_1, t_2, \ldots, t_m\}$ into the concept set $C = \{c_1, c_2, \ldots\}$, with the following steps, as Fig. 3 illustrates. The algorithm contains the following main steps.

1. Set up a confidence matrix $M(m \times m)$, where M_{ij} is a two-dimensional vector (s_{ij}, c_{ij}) in which s_{ij}, c_{ij} are the support and confidence of $t_i \rightarrow t_j$
2. Find M_{ij} with the largest c_{ij} in the confidence matrix M. Given a threshold δ, if $c_{ij} > \delta$, and $s_{ij} > \delta$, then merge these two clusters. Then, update M by replacing the two rows with a new row that describes then association between the merged cluster and the remaining clusters. The distance between the two clusters is given by the distance between the closet members. There are now $\mathrm{m} - 1$ clusters and $\mathrm{m} - 1$ rows in M.
3. Repeat the merge step until no more clusters can be merged.

In the clustering procedure, we initiate each term as a cluster. Then, the algorithm proceeds as a greedy one in order to sort association rules in descending order first by the confidence and then by support. The rules that are less than the given threshold δ are then abandoned. Every step chooses the highest ranking rules that have not been considered yet. If two terms in the rule do not belong to

```
Algorithm 1: Agglomerating Concepts from Input/Output
Inputs: a term set $I = \{t_1, t_2, ..., t_m\}$ from candidate web services input/output datatypes; a threshold $\delta$
Outputs: concept clusters $C = \{C_1, C_2, ...\}$

1.  FOR (i=1,m)
2.  $C_i = t_i$ //initialize the cluster with each term
3.  END FOR
4.  SETUP a Matrix $M(m \times m)$
5.  $M_{ij} \leftarrow < s_{ij}, c_{ij} >$ // $s_{ij}, c_{ij}$ are the support and confidence of $t_i \rightarrow t_j$
6.  REPEAT
7.  FOR EACH $M_{ij} \in M$
8.  IF $c_{ij} > \delta$ and $s_{ij} > \delta$
9.  Merge $C_i, C_j$ ;
10. Update M; // replacing the two rows with a new row that describes then association between the merged cluster and the remaining clusters.
11. UNTIL no clusters can be clustered
12. RETURN result clusters
```

Fig. 3. Clustering semantic concepts from data types.

the same cluster, the algorithm updates the matrix M by merging these two clusters. Note that the threshold δ controls the infrequent terms.

Particularly, it should be noted that these frequent and rare parameters should not be clustered. Such constraint promises that the improper words cannot be clustered, as they rarely occur from the statistic perspective. And the meaningless words, such as verbs and conjunctions, are removed in the clustering process, either. It guarantees that these strongly connected parameters in-between are clustered into concepts.

3.4 Improving Concepts Clustering

We argue that the clustering algorithm above is an "unsupervised bottom up" one that tries to cluster terms to concepts at a very coarse-grain level. The association rules are only based on the co-occurrence of terms and may lead to some inappropriate clusters. For example, the cluster related to concept of "weather" may contain terms *{temperature, humidity, wind, rain}*, and another cluster related to the concept of "address" contain terms *{zip, city, state, street}*. In our experiment, we found that some weather forecasting services often report temperature as well as city. According to the association rules, the confidence "*temperature→city*" is high, then the algorithm would merge the two clusters as a one. However, we know that these two clusters refer to different concepts, so they cannot be clustered.

In our approach, we employ the domain taxonomy for further evaluation. In the traditional information, the information content of a class or topic t is measured by the negative log likelihood $-\log\Pr[t]$. Hence, given a domain taxonomy D and two terms t_i, t_j, we evaluate their matching score MS by the **formula 1** as follows:

$$MatchingScore(t_i, t_j) = \frac{\log\Pr[D(t_i, t_j)]}{\log\Pr[t_i] + \log\Pr[t_j]},$$

TABLE 1
Example of Clustering Concepts

Concept	Term Set
Airline	{plane, time, date, flight, status, time, air,…}
Address	{city, street, zip, latitude, longitude, state, country,…}
Weather	{weather, termperature, humidity,wind, sun, rain, wet, dry, sky,pressure…}
Electronics	{model, manufacturer, hsize, vsize, power,..}

where t_i and t_j are two terms in the taxonomy, D[$t_i t_j$] is the lowest common ancestor in the domain taxonomy, and Pr[t] means the prior probability that t is classified under the taxonomy D.

Now, in the procedure of clustering, once merging two clusters, we take two steps. We first evaluate the association rules described in Algorithm 1. Then, we estimate the matching score of the related terms in the association rules. Formally, we assume two clusters I and J, and $t_i \in I, t_j \in J$, let $\rho \leftarrow \min(MS(t_i, I - t_i))$, if $MS(t_i, t_j) > \rho$, these two clusters cannot be merged. For example, although the confidence temperature→city is high, the matching score between temperature and city is lower than that of the temperature and the remaining terms, and these two clusters cannot be merged.

Obviously, the improvement by domain taxonomy is like "top-down" manner. Here, we do not discuss about the construction of domain taxonomy. There have been a lot of existing works in the IR community. In our practice, we find that the folksonomy[2] [33] meets our requirements very well. In those popular social networking Websites such as Flickrs [39] and del.icio.us [40], millions of user-annotated tags may represent the service's capabilities as well as user's requirements more accurately. We believe that it will generate more explicit domain taxonomy. In our experiment, we then employ the folksonomy generated from del.icio.us in [14].

3.5 Clustering Terms to Concepts

After the clustering approach mentioned above, we choose a subset in our data set (carefully introduced in Section 5) containing 506 different Web services with 3,328 operations and 4,936 inputs/outputs. With the clustering procedure, we finally get 2,076 terms and cluster 1,328 them into 263 concepts. We filter the overfrequent terms (which occur in at least 30 inputs/outputs) and the infrequent terms (which occur in at most three inputs/outputs). Moreover, with the help of the folksonomy, the clustering algorithm roughly assigns the terms to corresponding concepts such as **weather, finance, music, traffic, e-commerce, address**, and so on. Table 1 illustrates some example cluster results each of which holds more than six terms.

2. *Folksonomy* is the user-generated taxonomy, which describes the bottom-up classification terms that emerge from social tagging.

4 Discovering Homogeneous Web Service Community

In this section, we will introduce the discovery of the homogeneous Web services based on their similarities. We first predict the similarity by the multiple sources of evidence, and then, we propose a graph-based approach to find the similar operations as well as the potential composible operations.

4.1 Predicting the Similarity

In Section 3, we have clustered the concept as the baseline to measure the similarity for inputs/outputs. Now, we will compute similarity for the Web service operations. As defined in Section 3.1, an operation *op* is a three-tuple vector $op =< n_{op}, input, output >$, then given two operations op_i, op_j, we can determine the similarity by combining the similarity of each individual elements, respectively.

First, we estimate the similarity of the text description of operation and the Web services the operation belongs to (represented by N_w), it can be achieved by employing the traditional TF/IDF measurement.

Next, we estimate the similarity of the input and output by considering the underlying semantics the input/output parameters cover. Formally, we describe the input as a three-tuple vector $input =< n_{in}, C_i >$ (similarly, the output can be represented in the form of $output =< n_{out}, C_o >$), where n_{in} is the text description of input names and Ci is the concept that associates with n_{in}. Then, the similarity of *input* can be done in the following two steps:

- First, we evaluate the similarity of the descriptions of input names by TF/IDF.
- Second, we split n_{in} into a set of terms, just as described in Section 3. Note that we should filter the terms related to outputs (such as "ZipCode" in the input "CityNameByZipCode"). Then, we replace each term with its corresponding concepts, and then, use the TF/IDF measure.

The output can be processed in a similar fashion.

Now, we define the similarity between two operations op_i, op_j by the following formula:

$$Sim(op_i, op_j) = w_1 Sim(N_{wi}, N_{wj}) + w_2 Sim(n_{opi}, n_{opj}) + w_3 Sim(input_i, input_j) + w_4 Sim(output_i, output_j).$$

Here, $w_i (i = 1, 2, 3, 4)$ is the weight assigned to similarity of operation text description, input and output, $\Sigma w_i = 1$. Then, we define the two operations that are similar as follows:

Definition 4. *Given two operations op_i, op_j, a threshold ω, we claim that op_i, op_j are similar operations if $Sim(op_i, op_j) \geq \omega$.*

4.2 Constructing Service Aggregation Graph

We have completed the similarity measurement for operations. Now let's go back to the two service discovery types proposed in Section 2, that is, given a request of input and output, our approach returns:

Algorithm 2: Establishing Service Aggregation Graph
Input:
Operation set: OP, a given threshold ω
Output: A directed graph **SAG**

```
1.  InitG=null
2.  FOR (i=1;i ≤$\|OP\|$; i++)// $\|OP\|$ is the number of operations in the operation set
3.  IF $OPi \notin$ G
4.  THEN
5.  $Vi \leftarrow$ AddVertex (G, $OPi$ ) //add a vertex in current G
6.  END IF
7.  END FOR //now all vertexes have be created in the graph
//the following process establishes the edge between the vertexes
8.  FOR ( i=1;i ≤$\|OP\|$ ; i++)
9.  FOR ( j=1;i ≤$\|OP\|$ ; j++)
10. IF $Sim(output_i, input_j) \geq \omega$ //the output of op$_i$ can be accepted as the input of op$_j$
11. THEN
12. $e \leftarrow$ AddEdge ($V_i, V_j$ )//op$_i$ is the source vertex and op$_j$ is destination vertex
13. END IF
14. IF $Sim(input_i, output_j) \geq \omega$
15. THEN
16. $e \leftarrow$ AddEdge ($V_j, V_i$ )
17. END FOR
```

Fig. 4. Constructing the service aggregation graph.

- a list of similar single operations that can accept the request, and
- a sequence of operations that can be composed to fulfill the request.

In our approach, we process these two requirements within a directed graph $G =< V, E >$, in which:

- V is the vertex set. Each vertex in the graph G corresponds to an operation in our data set;
- E is the edge set. Each edge corresponds to the input or output of the vertex. We will discuss about the conditions to build the edges between vertexes later.

Fig. 4 illustrates the construction step for the graph. We name the established graph as "Service Aggregation Graph" (SAG) for it stores all operations. At the very beginning, we assign the graph to be empty and map each operation in the data set to vertexes iteratively. This step costs the complexity at the level of $O(n)$. Then, we establish the edges between the vertexes. For each vertex V_i, we check whether its corresponding output can be accepted as an input by a V_j in the graph, by computing the similarity of the parameters. If the similarity exceeds the given threshold ω, we then add an directed edge from V_i to V_j and assign the weight of this edge with similarity matching score. We also check if there exists a vertex V_j, whose output can be consumed by V_i as an input, in the similar manner. Obviously, these directed edges imply whether the corresponding operations can be composed sequentially (complied with the direction of the edge). The complexity of building up the edges in the graph is $O(n^2)$.

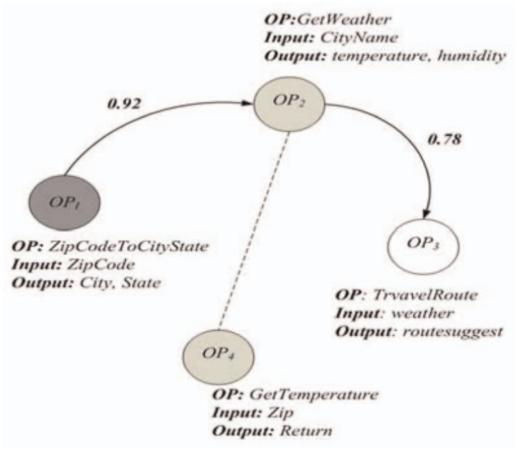


Fig. 5. Service aggregation graph.

Fig. 5 shows an example of SAG. Note that our approach also promises that there is no edge between vertexes representing similar operations, for example, Op_2, Op_4 in the example.

4.3 Service Discovery with the Service Aggregation Graph

After constructing the service graph, when the users submit a request $r =< I', O', \omega >$, where I' and O' represent the input and output data type the users desire and ω is threshold of similarity. We process r within the SAG.

In case of searching for the similar single operations, it just means that given a user request $r =< I', O', \omega >$, we should return a subset $OP' = \{op_1, op_2, \ldots, op_n\}, n \leq \|OP\|, \forall op_i \in OP', Sim(op_i, r) \geq \omega$.

Obviously, this problem can be transformed to traverse all vertexes that can meet the requirements. We use the general traverse algorithm to match the request with all vertexes in SAG, which costs the complexity at $O(|V|)$, where $|V|$ is the number of vertexes in the graph. Moreover, the operations are returned in the descending order of similarity matching scores.

In terms of discovery of the potential composible operations, the problem is transformed to discover a sequence of edge from a source vertex v_1 (which can perform I') to a destination vertex vn (which can perform O'). It means, given a request $r =< I', O', \omega >$, we should return a path $p = \{v_1 \rightarrow v_2, \ldots, \rightarrow v_n\}, v_i \in E$ in G, where $Sim(input_1, I') \geq \omega$ and $Sim(output_n, O') \geq \omega$.

We describe an algorithm based on dynamic programming to instead of the breadth-first traversal algorithm in [10] or AI-planning algorithm [21] for the composition-oriented service discovery. As shown in Fig. 6, our algorithm is based on the well-known *Dijkstra* shortest path algorithm [23], which can find the shortest path from the source vertex to the destination vertex.

The key idea of our algorithm lies in that it removes each edge in the top $i-1$ shortest path already which has been found to form a new subgraph and tries to find the shortest path in all the subgraphs by leveraging the Dijkstra's algorithm. The shortest one found in all the subgraphs is the ith shortest path. It can be easily proved that the our algorithm can find the $O(|V| \log |V| + |E| + K)$, where $|V|$

```
Algorithm 3: Discovering Top-K Composible Operations in SAG
Input:
request $r = <I', O', \omega>$, SAG
S: source vertex in the graph (similar to $I'$)
D: destination vertex in the graph (similar to $O'$)
K: the number of paths to consider
Output: ComOpLink: the array stores the top k shortest paths

1.  FOR (i=0;i ≤ K;i++)
2.    IF (i==0)
3.      THEN invoke Dijkstra (G,S,D, Path[0]) // invoke the Dijkstra
        algorithm to get the shortest path.
4.      RETURN Shortest Path in G
5.      $G \leftarrow G_0$ // assign G as the current $G_0$.
6.    ELSE
7.      FOR Each edge E in Path [i-1]
8.        remove E from $G_0$
9.        $G' \leftarrow G_0$ // get the new sub-graph G'
10.       invoke Dijkstra (G', S,D,TempPath)
11.       RETURN Shortest Path in G'
12.       Array CompOpLink ← Tempath, G'// Add the TempPath and
          G' to an array
13.     END DO
14.     Path[i] ← Shortest path in CompOpLink
15.     $G_0$ ← Path[i]'s corresponding graph // assign the path[i]'s
        corresponding graph to $G_0$
16.     remove Path[i] and $G_0$ from CompOpLink
17.   END IF
18. END DO
END
```

Fig. 6. Discovery of the composible operations.

and $|E|$ are the numbers of vertex and edge in the graph, respectively, K is the number of paths to be discovered.

In our approach, we consider only the weight on the edge (represented by the similarity matching score), as a constraint. Other constraints, e.g., the response time or cost for invoking the Web services, are not included. In fact, the *Multiple Constraints Shortest Path (MCSP)* is an NP-complete problem, and the solution is out of the scope of this paper.

5 Experimental Evaluation

In this section, we will set up a series of experiment to evaluate our approach. The experiments contain comparing the precision/recall of our approach with that of some existing work, the precision/recall in different domains, and the impact on the single operations and composible operations.

5.1 Web Service Corpus and Data Set Preparation

In [4], we have implemented our prototype, named as ***ServicePool***, to discover the homogeneous services based on the approach described above. In the ServicePool, we implement a crawler to download existing "real-world" WSDL resources from some well-known service registry sites [35], [36], [37], [38] and use google filetype search (e.g., filetype: wsdl). Overall, we download 1,084 wsdl files in total. We then removed the redundant ones with an MD5 algorithm, and finally, get a subset containing 506 different Web services with 3,328 operations and 4,976 inputs/outputs. Then, we store the operations into the SAG, finally, get 3,328 vertexes and 14,872 edges. Each vertex has 4.46 edges.

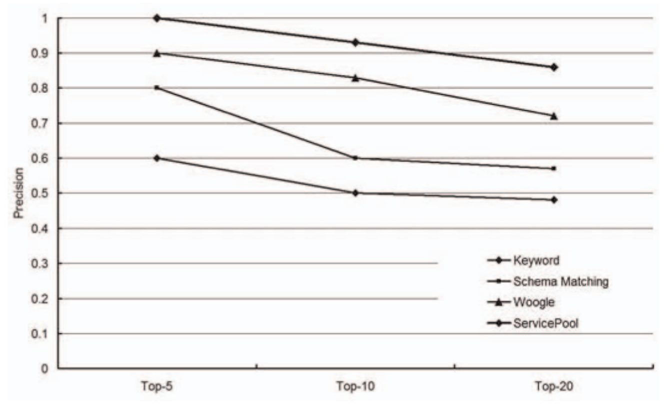


Fig. 7. Top-K precision comparison.

Given the users' request, ServicePool can return two types of results: the operations with similar inputs and outputs, and the sequentially composible operations. Note that the returned results by ServicePool are evaluated by combing both the two types (for the composible operations, we consider each path as a result and just take the top 5 shortest path in the experiment). Considering the usual metrics employed in traditional IR communities, we evaluate our approach with the recall (r) and precision (p).

Definition 5. *The recall $r = A/A + B$, where A stands for the number of returned relevant operations, B stands for the number of lost relevant operations, and $A + B$ stands for the total number of relevant operations.*

Definition 6. *The precision $p = A/A + C$, where A stands for the number of returned relevant operations, C stands for the number of irrelevant operations, and $A + C$ stands for the total number of returned operations.*

As mentioned in [2], we also involve the $R - P$ curve, whose X-axis is recall and the Y-axis is precision. It is an important metrics to measure the search results: a good search should have a horizontal curve with a high precision value.

5.2 Precision and Recall Measurement

First, we are going to evaluate the efficiency of ServicePool by comparing it with some existing Web service discovery approaches. We implement three different ones: keyword-based search, schema-matching based on WSDL, and the woogle [2]. We evaluate the precision of top-5, top-10, and top-20 returned results. To avoid potential bias, we choose different Web services from five different domain categories and compute the average results.

Fig. 7 illustrates the top-k precision comparison. We can see that both ServicePool and Woogle can achieve satisfying results, as they measure the textual descriptions as well as similarity based on the underlying semantics. For top-5, top-10, and top-20 precisions, ServicePool reaches 100, 91, and 87 percent, respectively, obviously higher than the other three. Woogle is a little lower at the

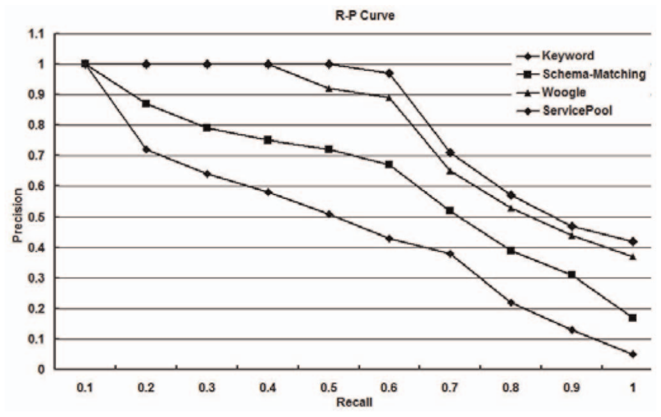


Fig. 8. R-P curve comparison.

level of top-5 and top-10, but the precision decreases at the level top-20. The reason is that woogle takes only consideration of the concept clustering by very slightly improvement (considering splitting and merging the cluster according to the cohesion score), without considering the domain taxonomy. Schema-matching has also very low precision, for it only considers the XML structure and the distance between nodes in the tree, while lack of the underlying semantics. Keyword-based approach is the lowest, for it only matches the keyword from the user's requests, and the results are coarse.

To measure the recall, we are interested in the number of relevant returned operations that can be returned within a given relevant operation set. With the results of concepts clustered in Section 3, we then manually classify 108 Web services with 411 operations into five different domain categories, and label whether the operations are relevant or not. To avoid bias, the person was a research student without previous experience with Web services. The person has the information as given in the registries, and was allowed to inspect the operation description. It guarantees that the different concepts can be conveyed. Finally, we collect a subset of 77 operations with 155 inputs and outputs.

Fig. 8 shows the R-P curve of the four approaches. Obviously, ServiePool and woogle beat the other two. The reason is that both of ServicePool try to attach the underlying semantic concepts to the input/output. Overall, ServicePool is a slightly better than woogle, for we take the folksonomy to evaluate the concepts to improve the clustering results and also study the composible operations rather than just single operations.

5.3 Precision in Different Domains

We also evaluate the precision considering domain taxonomy, we also set up an experiment with three domains, as shown in Fig. 9 ServiePool beats the other three approaches in each domain, since it filters the irrelevant concepts. Another interesting phenomenon is that, the precision in weather domain is a little higher than the other two.

We observe that, the operations related to weather takes a high percentage (17 percent) in our corpus, and provide more terms to cluster semantic concepts from the input/output data types. Another reason is that, we employ the folksonomy in Del.icio.us by the approach in

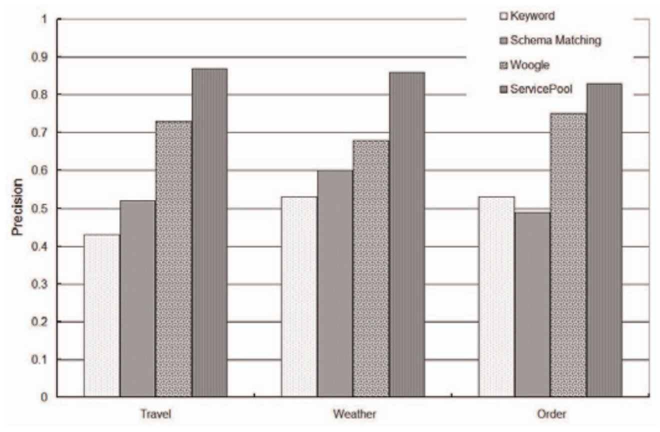


Fig. 9. Precision comparison in different domains.

[14] to evaluate the matching score, and investigate the annotations more than the other two. It also shows the evidence that domain taxonomy impacts the efficiency to concepts clustering.

5.4 Single Operation and Composible Operations

Finally, we are also interested in the effect on the two types ServicePool provided. So, we evaluate the precision and recall by comparing:

- *OpOnly:* just return the single operations can meet the requirements, and
- *Op&Comp:* return both the single operations and the composible operations.

We sort the returned results in a descending order according to the similarity matching scores to the request. As shown in Fig. 10, for the top-5, 10, and 20 returned results, *Op&Comp* can improve the precision at about 2, 5, and 14 percent, respectively, rather than *OpOnly*. It demonstrates that with the expected number of returned results increasing, the number of single operations decreases. In terms of recall, *OpOnly* is 97, 82, and 74 percent, respectively, as shown in Fig. 11. By considering the composible operations, *Op&Comp* can improve the recall beyond *OpOnly* at about 2, 11, and 12 percent, respectively. As ServicePool considers combing the single operations and

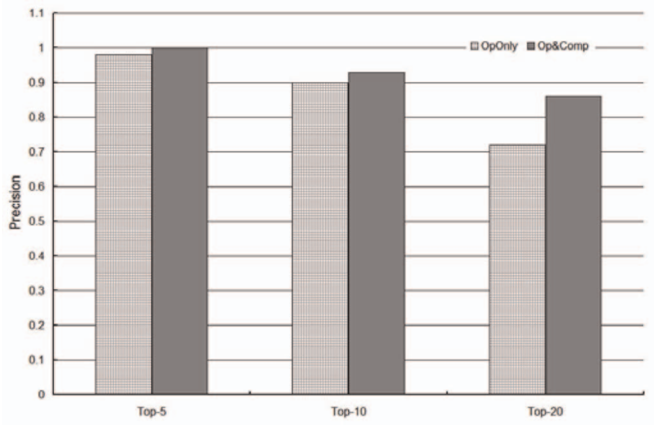


Fig. 10. Precision comparison between single operation and composible operations.

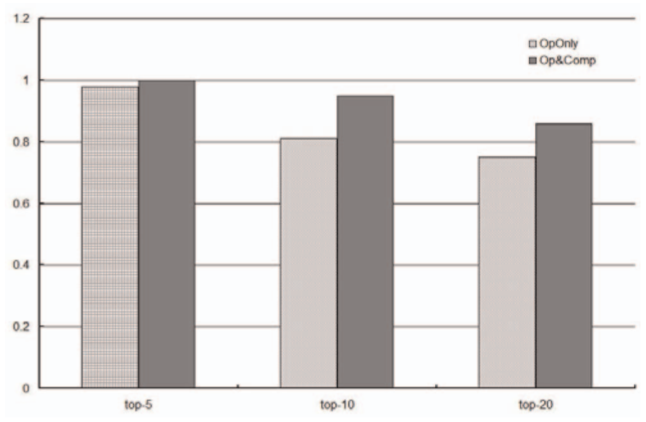


Fig. 11. Recall comparison between single operation and composible operations.

composible operations, it can improve both precision and recall significantly.

6 Prototype Implementation

We have implemented a prototype to demonstrate our ideas. In this section, we briefly describe the architecture of the prototype and the usability. The detail design can be retrieved in our previous work [4], [5].

6.1 Prototype Architecture

Fig. 12 describes the overall prototype architecture of ServciePool, which includes several components as well as the corresponding activities.

We assume that the service providers can publish their Web services on the Internet as usual. Then, we employ a *crawler* to retrieve the WSDL files from the service registries and store them into a local database as the metadata. The component *Metadata Parser* analyzes each WSDL file, filter the irrelevant information, and retrieve terms from the input/output data types. Using the similarity measurement

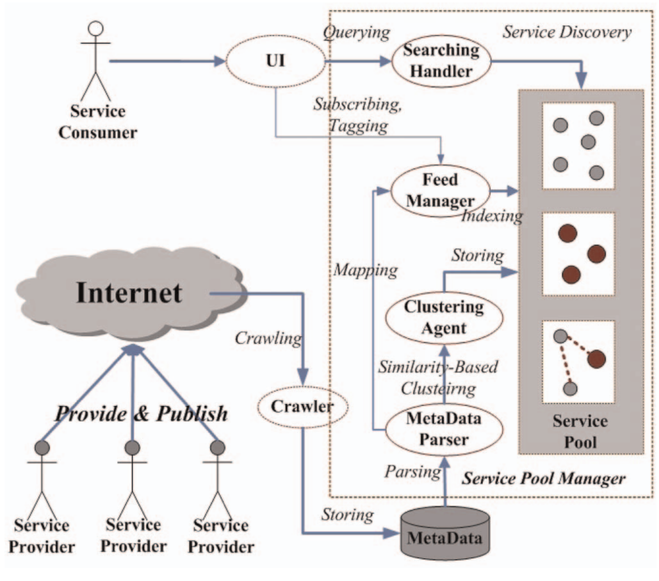


Fig. 12. Prototype architecture.

TABLE 2
Syntax Mapping from Atom Feed to WSDL

Atom	Semantics	Service Entry Binding
AtomCategory	Category associated to this entry	Any keyword tagged by its provider or retrieved from WSDL to represent core concept
AtomContent	Arbitrary markup	Service description from WSDL
AtomID	Unique identifier	Unique ID in ServicePool
AtomLink	Reference to from entry to resource	URL of WSDL or URL of service endpoint
AtomSummary	Short abstract of the feed	the metadata extracted from WSDL, including input/output, operation for indexing
AtomTitle	Entry title	Service name from WSDL

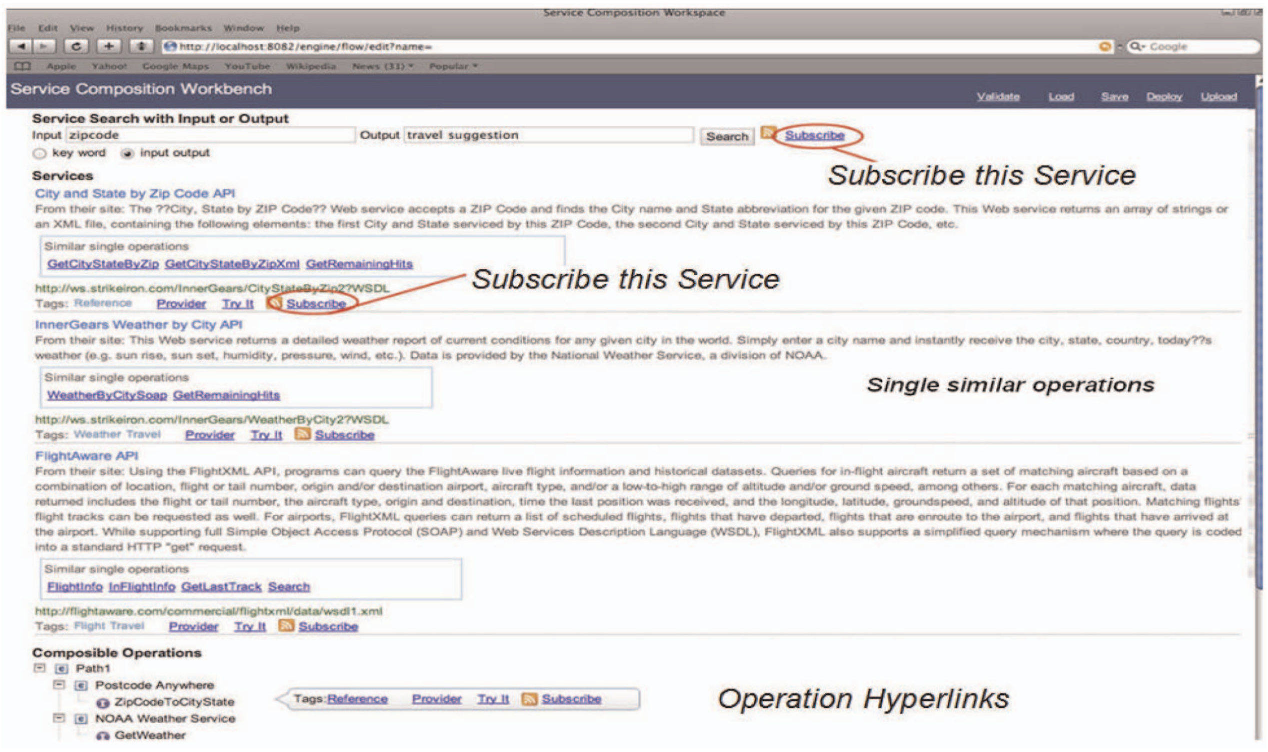
approach described in Section 3, the *Clustering Agent* clusters the terms into concepts, and then, finds the corresponding Web operations as well as the potentially composible ones. These operations are maintained in a graph, so we need to make index of them with the Web services they belong to.

As discussed before, considering the usability of the users, we adopt the Atom feeds (instead of popular RSS, for Atom is more suitable to process structured XML than RSS), which is widely used for news aggregation and subscription. In ServicePool, the *Feed Manager* component attaches the metadata to the Atom feeds and indexes the feeds to the corresponding Web services by using the standard Atom publish protocol.

Here, it is necessary to make the binding between the Atom feeds and WSDL, to ensure the consistency with the original Atom syntax and semantics. To the best of knowledge, in [20] and [24], the authors explicitly integrated atom feeds to a service description, including mapping the metadata of WSDL to the Atom feeds and the feed publish protocol over HTTP. This provides us very useful information in our implementation. We briefly describe the main design ideas.

Due to the functionalities that ServicePool provides, we adopt two feed types. The first one is the Web service entry, which is used to subscribe exactly one Web service. This element represents Web service by binding the metadata of the Web service to an Atom feed. We list the some core mappings in Table 2. For example, the "*AtomSummary*" is attached by the useful metadata from WSDL, including the textual description of input/output and operation, which can be used for indexing the Web services.

The second feed type is the topic feed, which is used for subscribing a homogeneous Web services (exactly their operations). The topic feed aggregates a set of homogeneous Web services into a group. The topic feed contains the metadata for a list of Web services, each of which corresponds to a service entry. The relationship between service topic feed and service entry is consistent with that defined in Atom specification. Service entry and topic feed correspond to the generic Atom feed and entry. Each entry in the feed is mapped to a single Web services under a particular topic.


Service Composition Workspace
File Edit View History Bookmarks Window Help
http://localhost:8082/engine/flow/edit?name=
Apple Yahoo! Google Maps YouTube Wikipedia News (31) Popular
Service Composition Workbench
Validate Load Save Deploy Upload
Search
Subscribe
Subscribe this Service
Subscribe this Service
Single similar operations
Operation Hyperlinks

Service Search with Input or Output

Input zipcode Output travel suggestion

key word [✓ input output]

Services

City and State by Zip Code API

From their site: The ??City, State by ZIP Code?? Web service accepts a ZIP Code and finds the City name and State abbreviation for the given ZIP code. This Web service returns an array of strings or an XML file, containing the following elements: the first City and State serviced by this ZIP Code, the second City and State serviced by this ZIP Code, etc.

Similar single operations
GetCityStateByZip GetCityStateByZipXml GetRemainingHits

http://ws.strikeiron.com/InnerGears/CityStateByZip2?WSDL
Tags: Reference Provider Try It Subscribe

InnerGears Weather by City API

From their site: This Web service returns a detailed weather report of current conditions for any given city in the world. Simply enter a city name and instantly receive the city, state, country, today??s weather (e.g. sun rise, sun set, humidity, pressure, wind, etc.). Data is provided by the National Weather Service, a division of NOAA.

Similar single operations
WeatherByCitySoap GetRemainingHits

http://ws.strikeiron.com/InnerGears/WeatherByCity2?WSDL
Tags: Weather Travel Provider Try It Subscribe

FlightAware API

From their site: Using the FlightXML API, programs can query the FlightAware live flight information and historical datasets. Queries for in-flight aircraft return a set of matching aircraft based on a combination of location, flight or tail number, origin and/or destination airport, aircraft type, and/or a low-to-high range of altitude and/or ground speed, among others. For each matching aircraft, data returned includes the flight or tail number, the aircraft type, origin and destination, time the last position was received, and the longitude, latitude, groundspeed, and altitude of that position. Matching flights' flight tracks can be requested as well. For airports, FlightXML queries can return a list of scheduled flights, flights that have departed, flights that are enroute to the airport, and flights that have arrived at the airport. While supporting full Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL), FlightXML also supports a simplified query mechanism where the query is coded into a standard HTTP "get" request.

Similar single operations
FlightInfo InFlightInfo GetLastTrack Search

http://flightaware.com/commercial/flightxml/data/wsdl1.xml
Tags: Flight Travel Provider Try It Subscribe

Composible Operations

- Path1
  - Postcode Anywhere
    - ZipCodeToCityState
  - NOAA Weather Service
    - GetWeather

Tags:Reference Provider Try It Subscribe

Fig. 13. User interface.

In the topic feed, the term *topic* is not the concept of WS-Topics [28], where topics are used to organize and categorize a set of notification messages [20]. In ServicePool, a topic is a list of some arbitrary keywords requested or tagged by the service consumers when searching the services. Hence, the topic definition is informal. For example, once the consumers search the weather forecasting services, they can tag the topic with any terms as they wish, e.g., "weather report," "rains," etc. Moreover, we even allow the users to create their own feeds when using ServicePool. In our opinion, these features are very important to improve the usability. First, although ServicePool aggregates a set of homogeneous services, we do not mandate the way they are organized. When the service consumers use ServicePool, the users can organize these services in their own way, instead of the complex and inflexible category in UDDI. It lowers the entry barrier for them to use Web services. Second, in the famous social networking sites such as flickr and del.icio.us, the personalized annotations are widely used by its numerous users. We believe that once the users feel something easy enough, they would prefer it. Actually, the success of recent social networking sites (such as Flickrs and del.icio.us) has proved that it is worth making things simple enough. Third, the user-annotated tags may more correctly represent the user's requirements with rich semantics. As we discussed in Section 3, we employ the folksonomy to improve the concept clustering. We believe that with the feedback of the service consumers, the efficiency of discovery will be promoted. Hence, our work not only alleviates the service from time-consuming and tedious service discovery work, but also reflects their importance in the user-centric Web environment.

6.2 User Interface

Finally, the service consumers can search, subscribe, and tag the homogeneous Web services within their Web browser. Fig. 13 shows the user interface in our prototype. The users can type in their desired input and output names and get the returned results. To make it more clearly, we classify the similar single operations and composible operations into two parts. The similar operations are highlighted as well as their inputs and outputs, in the description of the Web services they belong to. The composible operations are listed in a tree, where the upper node's output can be accepted as input by the lower ones. The returned results are sorted by their similarity against the requests.

Besides searching, the users are also able to subscribe both the ServicePool and the individual Web services like news or blogs by means of the atom feeds. As all the discovered services are registered on ServicePool server, the users can track the changes of subscribed feeds. Moreover, it should be noted that we even design a set of algorithms, which can plan a new order of services within ServicePool against the users' preferences, such as response time, price, and other QoS attributes [5], [7]. Hence, even if there are some changes, the ServicePool will send notification and delivers the latest discovered services to the users.

The users are allowed to give their feedbacks by attaching annotations. We argue that it is very useful feature for enabling the users to participate in the service discovery. Adopting these user-generated annotations will significantly enrich the service semantics and improve the service discovery.

7 Related Work

The Web service discovery is a hot research topic in the past a few years. Zhang et al. indicate that the service discovery and composition play the crucial role in the area of services computing [1]. To the service consumers, finding similar Web services and aggregating them in a universal access channel is a key requirement. There are some important research topics related to this issue. We classify the current discovery approaches into two categories: the syntactic-based discovery, which involves the techniques of UDDI-based search, text document search, schema matching, and software component matching; and semantic-based discovery, which is mainly based on ontology.

UDDI-based search. In the initial Web services architecture, UDDI works as the broker to register Web services into corresponding categories. However, to the best of our knowledge, the public UDDI never works as expected. In January 2006, the shutdown of UDDI Business Registry (UBR) operated collaboratively by Microsoft, SAP, and IBM has confirmed the intrinsic problem of the Internet-scale registry-based service discovery. The core reason of public UBR's failure is that the registry-based mechanism is "too complex" for the consumers. UDDI is mainly based on keyword search, which may bring several irrelevant results so that the consumers have to do the "view-select-request" process several times. It is too overwhelming for the consumers to simply get their desired services. Moreover, once the discovered services are no longer available, the discovery process has to be restarted. Thus, we cannot expect that these service consumers can utilize UDDI for service provisioning. The fact of UBR's shutdown has demonstrated that the "Internet-scale" public UDDI cannot be adopted by the huge number of Internet users. In our work, we investigate the service discovery problem from the service consumer's perspective and propose an approach to clustering the homogeneous Web services. It alleviates the consumers from tedious and time-consuming discovery step. With a much easier and universal channel (RSS/Atom) for the service consumers, they are able to subscribe and organize Web services just like Web pages, and track the updates and changes by means of service feeds.

Text document search. As Web services are specified in an XML document with an accessible URL, the keyword-based text document search is an intuitive approach. In IR community, document matching and classification is a long-standing topic and widely use in most search engines. Due to the fact that the great success of search engines promotes the Web-related search very much, it might be a natural idea to employ the current search techniques for similar Web service discovery. However, most of current information retrieval models are designed for Web pages crawlers and may not work well for Web service discovery due to some key reasons. First, Web pages may contain long textual information. However, Web services have very brief syntactic descriptions (from WSDL files). The lack of textual information makes keyword-based search models unable to filter irrelevant search results, and therefore, become very primitive means for effectively discovering Web services. Second, Web pages primarily contain plain text structures that allow search engines to take advantages of information retrieval models like TF/IDF. However, Web services contain much more complex structure with very little text descriptions provided either on UBRs or service interfaces. It then makes the dependency on information basic retrieval techniques very infeasible. Third, Web pages are described using standard HTML with predefined set of tags. However, Web service definitions are not fully standard as they are developed by independent vendors. Web service interface information such as message names, operation, and parameter names within Web services can vary significantly which makes the finding of any trends, relationships, or patterns within them very difficult and requires excessive domain knowledge in XML schemas and namespaces [3]. Therefore, the current text document search approaches are insufficient in the Web service context.

Schema matching. In the database community, the problem of automatically matching schemas investigates the clues of underlying semantics from the schema structure and suggests the matches based on them [18], [19]. In the Web service discovery, schema matching is also employed. In [17], the authors proposed an approach to measuring the similarity between two Web services based on their Tree Edit Distance. However, we argue that there is a big obstacle to apply schema matching to Web service discovery: the operations in a Web service are typically much more loosely related to each other than the tables defined in a schema, and each Web service in isolation has much less information than a schema. Hence, it will be difficult to retrieve the underlying semantics from the schema of WSDL. In our approach, we consider more precise semantic matching by employing some domain taxonomy, such as folksonomy from social annotations, and promise a more efficient results.

Software component matching. From the software perspective, Web services are autonomous software components that are accessible via standard interface specifications and protocols. Generally, the retrieval for software components is based on their signatures (structural) matching and specification (behavioral) matching [16]. However, these techniques employed there require analysis of data types and pre-/postconditions. In current WSDL files, the corresponding information is not available. To the best of our knowledge, the WSDL V 2.0 will adopt the semantic annotations for the data types and specifications.

Ontology-based discovery. A very important direction of current Web services research is the semantic Web services group, such as OWL-S [24], WSDL-S [25], and SWSO [27]. The initiative aim of semantic Web services is to make complementary for the Web service semantics beyond WSDL. For example, the OWL-S is proposed as the semantic markup for Web services. The OWL-S proposes well-defined structure including Service Profile, Service Model, and Service Grounding to represent the semantic information including Input, Output, Process, and Effect (IOPE). These semantic annotations specify the Web services at the semantic level. As summarized in [10] and [10], based on OWL-S, there have been several existing works on Web service discovery, such as automatic matching and selection [12], or even dynamic binding based on agent reasoning [13]. These efforts presume prebuilt ontology, thereby can regarded as "top-down" service discovery approach. However, constructing an ontology as a semantic backbone for a large number distributed Web services is really not easy. First, different organizations, people, and applications may have different views on what exists in the services and this leads to the difficulty of the construction of a commitment to a common ontology. Even if the consensus of a common ontology is achieved, it may not be able to catch the fast pace of change of the targeted Web service or the change of users' vocabularies in their applications. Second, using ontology for manual annotation requires that the annotator have some skills in ontology engineering which is a quite high requirement for normal consumers. In the Internet-scale environment, it is not feasible to mandate the Web service community to agree upon a few standards stipulating the way they should view the world. Actually, to the best of our knowledge, there are very few real-world Web services described by OWL-S.

Our work tries to retrieve the underlying semantic concepts from service metadata, and can be viewed as a "bottom-up" one. Experimental results on real-world Web services evaluate its efficiency. It should be noted that we do not mean to completely exclude the ontology. As we discussed in Section 3, we acknowledge that user-annotated taxonomy may be more accurately represent the service's capabilities as well as consumer's requirements. We believe that it will generate more explicit semantic categorization. In fact, tags are currently very popular descriptions, and can be obtained from those social networking Websites such as del.icio.us and flickr. Certainly, to realize the descriptions of service by tags, it still requires slight and more careful and consistent design, which is beyond the scope of this paper. In our opinion, we prefer relying on lightweight semantic metadata annotations by making use of tags, folksonomies, and simple taxonomies to describe the semantics of services [14]. The use of tag-based descriptions greatly simplifies the users, compared to the much heavier ontology-based approaches, such as OWL-S. To the best of our knowledge, we have found that some recent works [21], [22] have tried to attach the folksonomy to the service metadata and proposed AI-planning approach for autonomous discovery and composition of the Web services.

Another important related work worth mentioning is Woogle [2], a Web service search engine developed by the University of Washington. Woogle employs an unsupervised approach to retrieve the underlying semantics from WSDL and measure the similarity between operations and input/output. Similar to woogle, our approach adopts the semantic clustering algorithm to generate the meaningful concepts. As the concepts clustering results significantly impact the similarity measurement, we need to consider some improvement. Woogle provides a technique to split and merge the clusters by considering the cohesion and correlation. It can remove some terms and improve the clustering results. However, we notice that since the Web services are developed by independent providers, the parameter naming heavily relies on the developer's personal whim. Once a term in cluster A is associated with more than half of the terms in the cluster B, these two clusters will be merged by such technique. Therefore, in the phase of removing the noise terms, we apply the matching score by employing some taxonomy (use social folksonomy in our experiment), while woogle just processes by measuring the co-occurrence-based association rules. From experimental evaluation, we can see the our improvement beyond woogle. Another difference is that we establish a Graph-Based algorithm to find similar operations as well as the potentially composible operations and evaluate the efficiency.

8 Concluding Remarks and Future Work

In the past a few years, the Web communities have undergone a radical change to the "user-centric" environment where more and more users are able to directly participate in the Web-based computing [8]. Such change makes the numerous Internet users to leverage the Web for their own benefits and profits [20]. The emerging technologies (such as Web 2.0) and popular Web sites (such as flickr and del.icio.us) have met these users' requirements on easy participation, rich users interaction, and community-based collaboration.

In such a user-centric environment, it is undoubtedly convincing that the services computing paradigm can bring more and more benefits and profits to the users. This paper attempts to propose an efficient service discovery approach from the service consumer's perspective. After outlining the discovery requirements, we argue that the service consumers do prefer attaining the similar Web services and the potentially composible Web services, according to their desired inputs and outputs. These services should be organized in a universal access channel, instead of enforcing the users to search and view them individually. We call these services as "homogeneous" Web service community. Moreover, in the user-centric Web environment, the users may want to subscribe these services as RSS/Atom feeds, which is much easier than using UDDI. Based on the analysis, we propose our approach, which is named as ServicePool. To realize our approach, we then go to the problem of searching the similar service operation as well as the potentially composible ones. The basic idea is to measure the similarity between services by combing multiple sources of evidences. We investigate the structure of service description files (WSDL) and propose an algorithm to cluster the parameters (input/output) into the meaningful concepts. These concepts imply the underlying semantics of the Web services, and are leveraged to determine the similarity of inputs/outputs of the Web service operations.

After modeling the basis of similarity, we propose a graph-based algorithm to find the homogeneous Web services. All operations as well as their inputs/outputs are stored into a directed graph (SAG). Each operation is treated as a vertex in the graph, and the directed edge between the vertex is the representation of potentially composible operation sequences. Then, the user's requests are processed by searching the suitable vertex (for single similar operations) and the shortest path (for composible operations). For evaluation, we describe a series of experiments that contain comparing the precision/recall of our approach with that of some existing work, the precision/recall in different domains, and the impact on the single operations and composible operations. We analyze each experiment, and the results demonstrate the efficiency of our approach.

Finally, we give the design of our prototype supporting the idea. Within an intuitive search, the users can easily find their desired Web services including single similar operations and potentially composible ones. Moreover, to facilitate the consumers to track the changes of the Web services, we particularly support publishing both ServicePool and Web services as Atom feeds. Then, the users are able to subscribe the services as regular Web pages and annotate their own tags. The prototype not only alleviates the consumers from tedious service discovery work, but also reduces their entry barrier in the user-centric Web environment.

In terms of the future work, we aim to achieve the following goals. First, as shown in our prototype, we allow the users to annotate their own tags to describe the semantics. We are interested in how the user-annotated data can be utilized for further discovery. In this paper, we employ the taxonomy from de.licio.us, but it may be too general. As these annotations are service-specific, we believe that they may improve the similarity measurement. Second, in our current implementation, we only support the users to search by typing "input" and "output." Due to the fact that all the services in ServicePool can be published with Atom feeds, we plan to enable the feed-based search, which may be more easy-of-use for the Internet users. An assumptive scenario is that we transfer the Web services into feeds and publish these feeds as a "tag cloud." Then, the users can view the tag cloud to find the most frequently used tags (may reflect his requirement), click it and get the tags which can be composed together. Such manner is just like the "data mashup" like Yahoo! Pipes [34], which is a very popular style for service composition. As the best as we have known, there have already been some works [21], [22] considering the similar ideas. Finally, ServicePool is an aggregator for a set of services with similar functionalities, then we argue that it can act as a service "RAID" so as to improve the quality of service composition. It means that the ServicePool itself is a Web service that can process function-similar requests with different QoS. It then selects the best-of-breed candidates according to the QoS requirements and the users do not have to go to the details of every service. Once the current service is unavailable, ServicePool can transparently switch to another candidate. Certainly, there are some key techniques such as QoS-aware discovery, dynamic service switching, and state replication. Actually, in [7], we have made the first attempt to improve the dependability.

Acknowledgments

The authors wish to appreciate Professor Zhi Jin and Mr. Pei Wen from the Institute of Software, EECS, Peking University. This work was supported by the National Basic Research Program (973) of China under Grant No. 2009CB320703, the National Natural Science Foundation of China under Grant No. 60821003 and 60873060, and the EU Seventh Framework Programme under Grant No. 231167.

References

[1] L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*. Springer and Tsinghua Univ. Press, July 2007.

[2] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity Search for Web Services," *Proc. 30th Very Large DataBase Conf. (VLDB '04)*, 2004.

[3] C. Petrie, "The Myth of Open Web Services: The Rise of the Service Parks," *IEEE Internet Computing*, vol. 12, no. 3, pp. 93-95, May/June 2008.

[4] X. Liu, G. Huang, and H. Mei, "Consumer-Centric Service Aggregation: The Method and Framework," *J. Software*, vol. 18, no. 8, pp. 1883-1895, Aug. 2007.

[5] X. Liu, L. Zhou, G. Huang, and H. Mei, "Consumer-Centric Web Service Discovery and Subscription," *Proc. Int'l Conf. e-Business Eng. (ICEBE '07)*, pp. 543-550, 2007.

[6] X. Liu, L. Zhou, G. Huang, and H. Mei, "Towards a ServicePool Based Approach for QoS-Aware Web Services Discovery and Subscription," *Proc. ACM 16th Int'l Conf. World Wide Web (WWW '07)*, pp. 1253-1254, May 2007.

[7] G. Huang, X. Liu, and H. Mei, "SOAR: Towards Dependable Service-Oriented Architecture via Reflective Middleware," *Int'l J. Simulation and Process Modeling*, vol. 3, nos. 1/2, pp. 55-65, 2007.

[8] L.-J. Zhang, S. Ericksen, and J. Roy, "A Web 2.0 Tune-Up," *IT Professional*, vol. 9, no. 3, p. 9, May/June 2007.

[9] E. Al-Masri and Q.H. Mahmou, "Investigating Web Services on the World Wide Web," *Proc. 17th World Wide Web Conf.*, pp. 795-804, Apr. 2008.

[10] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, and F. Toumani, "On Automating Web Service Discovery," *VLDB J.*, vol 14, no. 11, pp. 84-96, 2004.

[11] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," *Proc. Int'l Workshop Semantic Web Services and Web Process Composition*, pp. 43-54, 2004.

[12] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX," *Proc. Fifth Int'l Joint Conf. Autonomous Agents and Multiagent Systems*, pp. 915-922, 2006.

[13] E.M. Maximilien and M.P. Singh, "A Framework and Ontology for Dynamic Web Services Selection," *IEEE Internet Computing*, vol. 8, no. 5, pp. 84-93, Sept./Oct. 2004.

[14] M. Zhou, S. Bao, X. Wu, and Y. Yu, "An Unsupervised Model for Exploring Hierarchical Semantics from Social Annotations," *Proc. Sixth Int'l Semantic Web Conf.*, Nov. 2007.

[15] S. Thummalapenta and T. Xie, "PARSEWeb: A Programmer Assistant for Reusing Open Source Code on the Web," *Proc. 22nd IEEE/ACM Int'l Conf. Automated Software Eng. (ASE '07)*, pp. 204-213, Nov. 2007.

[16] A.M. Zaremski and J.M. Wing, "Specification Matching of Software Components," *ACM Trans. Software Eng. and Methodology*, vol. 6, pp. 333-369, 1997.

[17] Y. Hao and Y. Zhang, "Web Services Discovery Based on Schema Matching," *Proc. 13th Australian Computer Science Conf. (ACSC '07)*, pp. 107-113, Jan./Feb. 2007.

[18] E. Rahm and P.A. Bernstein, "A Survey on Approaches to Automatic Schema Matching," *VLDB J.*, vol. 10, no. 4, pp. 334-350, 2001.

[19] H.H. Do and E. Rahm, "COMA: A System for Flexible Combination of Schema Matching Approaches," *Proc. Very Large DataBase Conf. (VLDB '02)*, 2002.

[20] C. Wu and E. Chang, "Aligning with the Web: An Atom-Based Architecture for Web Services Discovery," *Service-Oriented Computing and Applications*, vol. 1, no. 2, pp. 97-116. June 2007.

[21] A.V. Riabov, E. Bouillet, M.D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful Search: Interactive Composition of Data Mashups," *Proc. 17th Int'l Conf. World Wide Web*, pp. 775-784, Apr. 2008.

[22] E. Bouillet, M. Feblowitz, H. Feng, Z. Liu, A. Ranganathan, and A. Riabov, "A Folksonomy-Based Model of Web Services for Discovery and Automatic Composition," *Proc. IEEE Int'l Conf. Services Computing (SCC '08)*, pp. 389-396, July 2008.

[23] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2001.

[24] J. Snell, *Advertise Web Services with Atom 1.0*, http://www-128.ibm.com/developerworks/webservices/library/ws-atomwas/, 2009.

[25] W3C, *OWL-S: Semantic Markup for Web Services*, http://www.w3.org/Submission/OWL-S/, Nov. 2004.

[26] W3C, *Web Service Semantics*, http://www.w3.org/Submission/WSDL-S/, 2009.

[27] DAML, *Semantic Web Services Ontology,* http://www.daml.org/services/swsf/1.0/swso, 2009.
[28] OASIS, *Web Services Topics,* http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf, 2009.
[29] W3C, *Web Services Addressing,* http://www.w3.org/TR/2005/WD-ws-addr-core-20050331/, 2009.
[30] Web 2.0, http://en.wikipedia.org/wiki/Web_2.0, 2009.
[31] RSS Specification V 2.0, http://en.wikipedia.org/wiki/RSS, 2009.
[32] Atom Specification V 1.0, http://en.wikipedia.org/wiki/ATOM, 2009.
[33] Wikipedia Folksonomy, http://en.wikipedia.org/wiki/Folksonomy, 2009.
[34] Yahoo! Pipes, http://pipes.yahoo.com, 2008.
[35] Xmethods, http://www.xmethods.com, 2009.
[36] Strikeiron, http://www.stikeiron.com, 2008.
[37] WebServiceX, http://www.webservicex.net, 2009.
[38] Esynaps, http://www.esynaps.com, 2009.
[39] Flickr, http://www.flickr.com/, 2008.
[40] Del.icio.us, http://www.delicious.com/, 2009.

Xuanzhe Liu is currently working toward the PhD degree at the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. His research interests are in the areas of software engineering, services computing, and social computing. For more information, please visit his homepage at http://www.sei.pku.edu.cn/~liuxzh.

Gang Huang received the PhD degree from the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. He is now an associate professor in the Institute of Software, Peking University. His research interests are in the areas of distributed computing with an emphasis on the construction and management of middleware and software engineering with an emphasis on component/service-based development and software architecture. He is a member of the IEEE. For more information, please visit his homepage at http://www.sei.pku.edu.cn/~huanggang.

Hong Mei received the PhD degree in computer science from Shanghai Jiao Tong University in 1992. He is a full professor in the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. His current research interests are in the areas of software engineering and software engineering environments, software reuse and software component technology, and distributed object technology. He is a member of the Expert Committee for Information Technology of the State 863 High-Tech Program, a chief scientist of the State 973 Fundamental Research Program, a consultant of Bell Labs Research China, the director of the Special Interest Group of Software Engineering of China Computer Federation (CCF), and a member of the editorial board of *Sciences in China (Series F)*, *ACTA ELECTRONICA SINICA*, the *Chinese Journal of Electronics*, the *Journal of Software*, the *Journal of Computer Science and Technology*, *Computer Research and Development*, *Natural Sciences in China*, and the *International Journal of Web Services Research* (JWSR). He is a senior member of the IEEE. For more information, please visit his homepage at http://www.sei.pku.edu.cn/meih/index_en.html.