Efficient IR-Style Search over Web Services

Yanan Hao[1], Jinli Cao[2], and Yanchun Zhang[1]

[1] School of Engineering and Science, Victoria University
P.O. Box 14428, Melbourne, VIC 8001, Australia
{haoyn,yzhang}@csm.vu.edu.au
[2] Department of Computer Science and Computer Engineering, La Trobe University
Bundoora, VIC 3086, Australia
j.cao@latrobe.edu.au

Abstract. In service-based systems, one of the most important problems is how to discover desired web services. In this paper, we propose a novel IR-Style mechanism for discovering and ranking web services automatically. In particular, we introduce the notion of preference degree for web services and then we define service relevance and service importance as two desired properties for measuring the preference degree. Furthermore, various algorithms are given for computing the relevance and importance of services, respectively. Experimental results show the proposed IR-style search strategy is efficient and practical.

1 Introduction

Service-Oriented Computing (SOC) is emerging as a new paradigm for developing distributed applications. Web service discovery, among the most fundamental elements of SOC, provides a way to combine basic web services into value added services to satisfy user needs. As the number of web services and Service Oriented Computing applications increases, there is a growing need for mechanisms for discovering services efficiently.

Web service discovery introduces many new challenges. First, current web service discovery methods are mostly based on the UDDI-registry. To find a service in UDDI, a user needs to browse the relevant UDDI category to locate relevant web services. Considering a large amount of service entries, this process is time consuming and frustrating. So, we need an effective mechanism for automatic web service discovery. Second, a user's requirement for desired web services may not always be precise and a service discovery mechanism can potentially return a large number of results to satisfy the user's requirement, especially when a large service repository is available. Consequently, an important requirement for web service discovery is to rank the discovered results so that the most relevant services appear first. Finally, a good web service discovery mechanism should also be able to assist users in selecting relevant services and compose with them. For example, a typical strategy would allow users to see the services first she can use to start composing her application. Consider the three services shown in Fig. 1. The second and the third services can process the order information for

P. van Eck, J. Gordijn, and R. Wieringa (Eds.): CAiSE 2009, LNCS 5565, pp. 305–318, 2009.

WS1: Web Service: CreateOrder Operation: *OrderBuilder* Input: UserID DataType: *int* Requirement DataType: *ItemList* **Output**: ProductsList DataType: *BuyingOrder*
WS2: Web Service: ProcessPayment Operation: *CheckoutOrder* **Input**: UserProducts DataType: *UserOrder* Output: PaymentConfirmation DataType: *bool*
WS3: Web Service: TransportOrder Operation: *ShippingOrder* **Input**: Cargo DataType: *Order* Output: PickupTime DataType: *TimeLimit*

Fig. 1. Sample web-service operations

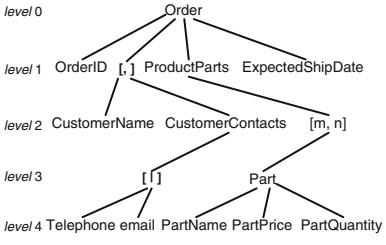

Fig. 2. XML schema tree of *Order* type

one transaction provided that a buyer's order has been generated; whereas the first service provides the buyer's order according to her requirement. Obviously, it is reasonable to say the first service is more important than the others since it contributes indispensable information for both of the other two services to be invoked. So, an ideal ranking strategy should put the first service on top. Also, as we can see, there are two links between the first and the other two services, in which the output of *CreateOrder* service, *BuyingOrder*, is also the input of both *ProcessPayment* service and *TransportOrder* service. This form of link potentially involves more web services and thus are particular useful in web service composition.

To address the problems above, in this paper we propose a novel IR-Style mechanism for discovering and ranking web services automatically, given a textual description of desired services. The contribution of the work reported here is summarized as follows:

1. We introduce the notion of *preference degree* for web services and then we define *service relevance* and *service importance* respectively as two desired properties for measuring the preference degree.

2. We design novel algorithms for computing the relevance and importance degree of services. Our algorithms take into account both textual and structural information of web services.
3. We define *service connectivity*, a novel metric to evaluate the importance of services. Meanwhile, we use our existing schema tree matching algorithm to measure the service connectivity.
4. We do various experiments to search for desired web services. Initial results show the proposed IR-style search strategy is efficient and practical.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the conception of preference degree for service ranking. Section 4 and section 5 present models and definitions for service relevance and service importance, followed by section 6, in which we present algorithms for ranking web services. In section 7 we describe our experimental evaluation. Section 8 gives some concluding remarks.

2 Related Work

Most approaches use text or structural matching to find similar web services for a given web service. The earlier technique tModel presents an abstract interface to enhance service matching process. But the tModel needs to be defined while authors publishing in UDDI [1]. In [2], the authors propose a SVD-Based algorithm to locate matched services for a given service. This algorithm uses characteristics of singular value decomposition to find relationships among services. But it only considers textual descriptions and can not reveal the semantic relationship between web services. Wang etc. [3] discover similar web services based on structure matching of data types in WSDL. The drawback is that simple structural matching may be invalid when two web-service operations have many similar substructures on data types. Woogle [4] develops a clustering algorithm to group names of parameters of web-service operations into semantically meaningful concepts. Then these concepts are used to measure similarity of web-service operations. However, it relies too much on names of parameters and does not deal with composition problem.

Recently, some methods have been proposed to annotate web services with additional semantic information. These annotations are used to match and compose services. For example, in [5] the authors extended DAML-S to support service specifications, including behavior specifications of operations; The Web Service Modeling Ontology (WSMO) [6] is a conceptual model for describing Web services semantically, and defines the four main aspects of semantic Web service, namely Ontologies, Web services, Goals and Mediators. However, most of existing web services currently use WSDL specifications, which do not contain semantics. Annotating the collection of services requires much effort, and it is infeasible in web service discovery. [7] formally defines a behavior model for web service by automata and logic formalisms. However, the behavior signature and query statements need to be constructed manually, which can be very hard for common users.

Some of our algorithms to be presented in this paper are related to keyword search in databases or Internet. For example, Discover [8,9] and DBXplore [10] operate on relational databases and facilitate information discovery on them by allowing users to issue keyword queries without any knowledge of the database schema; Google PageRank [11] uses the Internet's link structure as an indication of each web page's importance value. Also, our work is inspired by the work on XML schema. For instance, in [12] authors propose a syntactic approach to web service composition, given only the input-output schema types of web services available in their WSDL descriptions; [13] introduces the concept of schema summary and suggests importance and coverage as two relevant properties by which to judge the quality of a schema summary. In our previous work [14], we proposed a new schema matching algorithm for supporting web-service operations matching. The matching algorithm catches not only structures, but even better semantic information of schemas.

Other approaches include P2P-based service discovery [15], QoS-based discovery and ranking [16], service crawler [17]. Although these work provide ranking strategies, they overlook user's semantic preferences and can not identify the semantic relationship between services.

3 Desired Properties for Service Rank

Our goal is to find services in a more automatic and IR-style way, given a potentially partial specification of the desired service. We need an efficient mechanism to select the preferred services from available ones to satisfy the user's requirement. A natural idea is, firstly, to evaluate the user's preference for available services with respect to the textual service requirement, rank them according to the degree of preference, and then return the top services as search results. But, what makes a good service to the user? What does "preference degree" mean and how do we compute it? In [11], authors pointed that the final rank of a web page appearing in search results pages is determined by both the goodness of the match of the search terms on the page itself (*relevance* of the page) and this page's PageRank (*importance* of the page). Extending this idea to our context, we consider two factors for the user's preference degree for a service, in other words, the rank of the service. First, a good web service should be relevant to the user's requirement, i.e., to a certain extent, similar to the service requirement; Second, we should select services that are important. Intuitively, a service is important if it is employed by many other services in service composition; therefore, services that can be employed by as many as services are worth looking at.

Having seen what we consider to be desired properties for ranking web services, in the next two sections we will define the *service relevance* metric and *service importance* metric respectively, and then we calculate the user's preference degree for available services in reference to a textual description of desired web services provided by the user.

4 Service Relevance

Let q be a natural language description of the desired web services, $S = \{s_1, s_2, ..., s_k\}$ be the set of all available services published through UDDI; and $D = \{D_1, D_2, ..., D_k\}$ be a document collection containing WSDL specifications for all the services in S, where each WSDL document D_i corresponds to service s_i. Suppose there are N distinct words in D after a pre-processing step, including word stemming, removing stop words and expanding abbreviations and acronyms into the original forms. Applying the vector-space model to web services context, we describe each service s_i as an N-dimensional vector $\overrightarrow{D}_i$ containing all terms in its specification D_i, denoted as $\overrightarrow{D}_i = \{(t_1, w_{i1}), (t_2, w_{i2}), ..., (t_N, w_{iN})\}$, where each term is assigned a weight. A well-known weighting method is TF/IDF, namely the normalized *term frequency* (TF) and *inverse document-frequency* (IDF). Typically, the weight for each term t_j in document D_i is given by $w_{ij} = tf_{ij} \times idf_j = tf_{ij} \times \log(k/n_j)$, where k is the total number of available web services and n_j is the number of corresponding WSDL documents in which the term t_j appears. For more details, interested readers are referred to see [18].

Given the weighted vector $\overrightarrow{q}$ for the user's description of desired services and the weighted vector $\overrightarrow{D}_i$ for a service s_i's WSDL specification document, we adopt the cosine distance metric to compare their similarity. Formally, the *service relevance* can be defined as follows:

Definition 1. (`Service Relevance`). *The relevance of a service s_i, denoted as R_{si}, with respect to the user's natural language description of desired web services, written as q, is defined as:*

$$R_{s_i} = \frac{\sum_{k=1}^{N} w_{ik} \times w'_{ik}}{\sqrt{\sum_{k=1}^{N} (w_{ik})^2} \cdot \sqrt{\sum_{k=1}^{N} (w'_{ik})^2}} \tag{1}$$

where w_{ik} is the weight for term k in $\overrightarrow{s}_i$ and w'_{ik} is the weight for term k in $\overrightarrow{q}$. R_{s_i} ranges from 0 to 1. The higher score R_{s_i} is, the higher relevance service s_i has with respect to q, indicating a closer similarity between the user's description of request q and the available web service s_i.

5 Service Importance

A web service is in some way not different from a software component or module. Like in a software library, where different functions or modules have different level, not all services have equal importance in a web service repository. For example, consider the services in Fig. 1. Although all their WSDL descriptions contain terms provided by the user to search for desired services, most people would agree that service *CreateOrder* is more important than both the *ProcessPayment* service and the *TransportOrder* service, since the output of the first service, *BuyingOrder*, is also the input of the other two services, and thus a must

prerequisite for the other two services to be invoked. Based on this observation, we argue that a service can have great importance if there are many other services that employ it, or if there are some services that employ it and have a great importance. From this point of view, a service having low relevance may be more important than a service showing high relevance.

In order to reveal the nature of the importance of a service, we need to identify the relationships between the service and other services. Furthermore, we need an appropriate metric to capture how well a service can be used by other services. In the following parts, we describe models and algorithms to evaluate service importance; in particular, we show how to measure *connectivity* between two web-service operations based on schema matching, and how by connectivity we can achieve the importance of a web-service operation, which contributes to part of the importance of the service it belongs to.

5.1 Web-Service Operation Modeling

Definition 2. *A web service is a triple* $ws = (TpSet, MsgSet, OpSet)$*, where TpSet is a set of data types; MsgSet is a set of messages (parameters) conforming to the data types defined in TpSet;* $OpSet = \{op_i(input_i, output_i)|i = 1, 2, ..., n\}$ *is a set of operations, where* $input_i$ *and* $output_i$ *are parameters (messages) for exchanging data between web-service operations.*

Fig. 1 has given three web-service operations used as examples in this paper. According to definition 1, a web service can be briefly described as a set of operations.

Definition 3. *Each web-service operation is a multi-input-multi-output function of the form* $f : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$*, where* s_i *and* t_j *are data types in according with XML schema specification. We call f a* ***dependency*** *and* s_i/t_j *a* ***dependency attribute****.*

A dependency attribute can be a complex data type or a primitive data type. Complex data types, such as *BuyingOrder* and *UserOrder* in Fig. 1, define the structure, content, and semantics of parameters, whereas primitive data types, like *int* and *bool*, are typically too coarse to reflect semantic information. Since parameters usually can be regarded as data types, we can convert primitive data types to complex data types by replacing them with their corresponding parameters. For example, in Fig. 1 *bool* is converted into *PaymentConfirmation* type while *int* is converted into *UserID* type. Both *PaymentConfirmation* and *UserID* are considered as complex data types with semantics. Therefore, now each data type defined in a web-service operation can carry semantic meaning, according with XML schema specification at the same time.

5.2 Connectivity of Web Service Operations

As we can see, data types defined in web-service operations carry semantic information. Intuitively, we can consider two web-service operations, say A and

B, connected if the output data types/attributes of A is the same as the input data types/attributes of B, so service B could directly employ service A's output result and they can potentially collaborate in a user's web-service composition process. Obviously, however, requiring that A's output and B's input are the same so as to be connected is too strict and not practical in many cases. Generally, the connectivity relationship between two web-service operations can be defined formally as below:

Definition 4. (`Web-service Operation Connectivity`). *Given two web-service operations* $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$ *and* $op_2 : u_1, u_2, ..., u_l \rightarrow v_1, v_2, ..., v_k$, *Let* $X = \{t_1, t_2, ..., t_m\}$ *and* $Y = \{u_1, u_2, ..., u_l\}$. *The connectivity of* op_1 *with respect to* op_2 *can be measured as the similarity degree between* X *and* Y, *denoted as* $Con_{op_2 \rightarrow op_1} = sim(X, Y)$.

Service operation op_1 is said to be *connected* to op_2 If the connectivity degree $Con_{op_2 \rightarrow op_1}$ is greater than some threshold value $\lambda (0 < \lambda < 1)$. If op_1 has exactly the same output data type as op_2's input, we will have $Con_{op_2 \rightarrow op_1} = sim(X, Y) = 1$, indicating the highest possible degree of connectivity of op_1 regarding op_2. In this case, we say op_1 is *well connected* to op_2. On the contrary, if the output of op_1 is totally different from op_2's input, we will have $Con_{op_2 \rightarrow op_1} = sim(X, Y) = 0$, indicating the lowest possible degree of connectivity of op_1 regarding op_2.

As we have known, a data type used in web service operations presents a structure of schema tree, so X and Y are actually two groups of schema trees. Therefore, we can convert the problem of measuring connectivity between two web-service operations to the problem of schema tree matching. Section 6 will detail the algorithms for deriving the connectivity of a web service operation.

5.3 Importance of Web Service Operations

Based on the notion of connectivity, the web-service operation importance is given by an iterative equation as below, similar to the technique used in PageRank [11] algorithm:

Definition 5. (`Web-service Operation Importance`). *The importance of a web-service operation op, written as* I_{op}, *is calculated as the following iterative formula until convergence is reached:*

$$I_{op}^{r} = (1 - p) + p * \sum_{j \in F_{op_j}} Con_{op_j \rightarrow op} * I_{op_j}^{r-1} * 1/N_{op_j} \tag{2}$$

where $Con_{op_j \rightarrow op}$ is the connectivity degree of op with respect to service operation op_j; r denotes the number of iterations; F_{op_j} is the set of service operations connected by op_j; $N_{op_j} = \left|F_{op_j}\right|$ is the number of operations in F_{op_j}; and $0 \leq p \leq 1$ is a tuning parameter indicating how well the importance of a web service operation is affected by that of others. For all web-service operations,

the initial importance I_0 is set to $1/N$, where N is the total number of available web-service operations. The computing process of the iterative equation above is shown by the *CompImp* algorithm in section 6.2.

6 Algorithm for Ranking Web Services

We now turn to the main focus of this paper, which is efficiently ranking web services. Recall that in section 3, two factors are considered for the rank of a service: service relevance and service importance. Since service relevance has been discussed in section 4, now the key issue remaining is how to compute the importance of services. We start with computing the connectivity of web-service operations by our schema tree matching strategy, then iteratively compute the importance of operations by *CompImp* algorithm. Finally, we combine these two factors to achieve the final rank scores for all web-service operations.

6.1 Computing Connectivity Using Schema Tree Matching

In this section, we use our existing schema tree matching algorithm [14] to measure the connectivity of a web-service operation, which is also a key step for evaluating its importance.

Our schema matching algorithm is based on tree edit distance [19,20]. However, traditional tree edit distance methods have three shortcomings:

1. They consider all tree edit operations to have same unit distance.
2. They neglect semantic information carried by the labels of nodes.
3. They do not consider the node difference between tag nodes and constraint nodes, and assign each edit operation unit cost.

To overcome these shortcomings, we presented a new cost model to compute the cost of tree edit operation, by which the tree edit distance of two schema trees is achieved. The new cost model integrates weights of nodes and semantic connections between nodes. Let T_1, T_2 be two schema trees and let n, $node_1$ and $node_2$ be tree nodes. Formally, the cost model is defined as

$$cost(\rho) = \begin{cases} weight(n)/W(T_1,T_2), & \text{if} \rho = insert(n) \\ weight(n)/W(T_1,T_2), & \text{if} \rho = delete(n) \\ \alpha \times wd(node_1, node_2) & \text{if} \rho \text{ relabels} \\ +\beta \times sd(node_1, node_2) & node_1 \text{to } node_2 \end{cases} \tag{3}$$

where ρ indicates a tree edit operation. $weight(n)$ shows the weight of node n, which is defined in definition 5. $wd(node_1, node_2)$ and $sd(node_1, node_2)$ give the weight and semantic difference of $node_1$ and $node_2$, respectively. α and β are weights of wd and sd, satisfying $\alpha + \beta = 1$. $W(T_1, T_2)$ is defined as $W(T_1, T_2) = weight(T_1) + weight(T_2)$, where $weight(T_i)$ is the sum of all node weights of tree $T_i (i = 1, 2)$. $wd(node_1, node_2)$ is defined as

$$wd(node_1, node_2) = \frac{\|weight(node_1) - weight(node_2)\|}{W(T_1, T_2)} \tag{4}$$

where $node_1 \in T_1$ and $node_2 \in T_2$.

In equation 3, $weight(n)/W(T_1, T_2)$ explains the cost of inserting or deleting node n. For the relabel operation, both weight and semantics of $node_1$ and $node_2$ can be different, so we use the combination of weight and semantic difference as the relabel cost. All the costs are normalized by $W(T_1, T_2)$, i.e. the sum of all nodes weights of tree T_1 and T_2.

Definition 6. *Let $level(n)$ denote the level of node n in schema tree T. The weight of node n is defined by a weight function:*

$$weight(n) = 2^{depth(T)-level(n)} (\forall n \in T) \tag{5}$$

The weights of all nodes fall in the range of $[2, 2^{depth(T)}]$. Each weight reflects the importance of a node in schema tree T.

Having seen that traditional tree edit distance algorithms are not suitable for XML schema trees, three transformation rules: *split*, *merge* and *delete* are proposed to solve this problem. These rules are used to transform constraint nodes, specifically, sequence nodes, union nodes and multiplicity nodes to tag nodes. At the same time, the weights of nodes are reassigned. Fig. 3 gives an example of the three rules.

Note that the definition of complex types can be nested according to XML schema specification. Thus, given a schema tree, we apply the three transformation rules to its nodes level by level, from bottom to top. After the bottom-up transformation, schema tree T is converted into a new schema tree T^*. Each node n of T^* is a tag node, which contains a few words.

Our idea relies on a hypothesis that two co-occurrence words in a WSDL description tend to have same semantics. We exploit the co-occurrence of words in word bags to cluster them into meaningful concepts. To improve accuracy of semantic measurement, we first carry out the pre-processing step before words clustering, which has been done in the service relevance computation.

Then we use the agglomeration algorithm [21] to cluster words set $I = \{w_1, w_2, ..., w_m\}$ into concept set $C = \{C_1, C_2, ...\}$. There are three steps in the clustering process. It begins with each word forming its own cluster and gradually merges similar clusters.

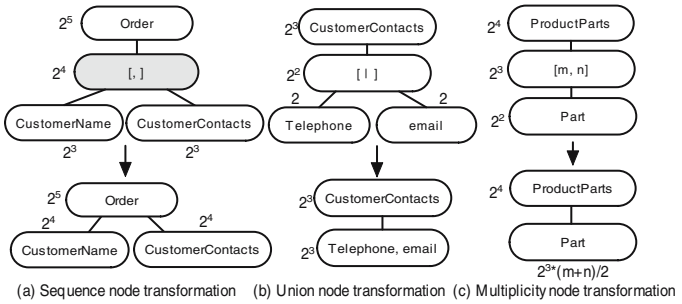

Fig. 3. XML Schema tree transformation

Finally, we get a set of concepts C. Each concept C_i consists a set of words $\{w_1, w_2, ...\}$. To compute semantic similarity between schema-tree nodes, we replace each word in tag nodes with its corresponding concept, and then use the TF/IDF measure. After schema-tree transformation and semantic similarity measure, the tree edit distance can be applied to match two XML schema trees by the new cost model.

Obtaining Connectivity. As it has been mentioned before, we use tree edit distance to match two schema trees. It is equivalent to finding the minimum cost mapping. Let M be a mapping between schema tree T_1 and T_2, let S be a subset of pairs $(i, j) \in M$ with distinct word bags. Let D be the set of nodes in T_1 that are not mapped by M, and I be the set of nodes in T_2 that are not mapped by M. The mapping cost is given by $C = Sp + Iq + Dr$, where p, q and r are the costs assigned to the relabel, insertion, and removal operations according to the cost model proposed in section 6.1.2. We call C the *match distance* between T_1 and T_2, denoted as $C = ED(T_1, T_2)$. Match distance reflects semantic similarity of two schema trees.

Now let us see how to compute the connectivity of a web-service operation.

Given two web-service operations $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$ and $op_2 : u_1, u_2, ..., u_l \rightarrow v_1, v_2, ..., v_k$, Let $X = \{t_1, t_2, ..., t_m\}$ and $Y = \{u_1, u_2, ..., u_l\}$. The connectivity of op_1 with respect to op_2 is $Con_{op_2 \rightarrow op_1} = sim(X, Y)$. To achieve $sim(X, Y)$, for each schema tree $\in X$, we find its corresponding schema tree $\in Y$ with the minimum match distance. We simply identify all possible matches between two lists of schema trees X and Y, and return the source-target correspondence that minimizes the overall match distance between the two lists. It does not depends on whether the number of schema trees is the same or not between X and Y. This process is illustrated by algorithm 1.

input : $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$
$op_2 : u_1, u_2, ..., u_l \rightarrow v_2, ..., v_k$
output: The connectivity of op_1 with respect to op_2

1 **for** $i \leftarrow 1$ **to** m **do**
2 | $S_i = min\{ED(t_i, u_j) | j = 1, 2, ..., l\}$;
3 **end**
4 $Con_{op_2 \rightarrow op_1} = \sum_{i=1}^{m} S_i$

Algorithm 1. Algorithm for computing web-service operation connectivity

6.2 The *CompImp* Algorithm

Based on the strategies proposed in section 5.3, we design the *CompImp* algorithm for automatically computing the importance of a set of given web-service operations $OP = \{op_1, op_2, ..., op_N\}$. The algorithm iteratively computes the importance values for all operations until convergence. It initializes the importance

```
input  : A set of web-service operations OP = {op_1, op_2, ..., op_N}
output: An array I[1 : N] to store the importance values of OP
1  foreach service operation op_i ∈ OP do
2  |  I_i^cur = 1/N;
3  |  convergence[1 : N]=false;
4  end
5  repeat
6  |  foreach service operation op_i ∈ OP do
7  |  |  calculate I_i^cur using equation 2;
8  |  |  if |I_i^new − I_i^cur|/I_i^cur ≤ c then
9  |  |  |  convergence[i] = true;
10 |  |  else
11 |  |  |  convergence[i] = false;
12 |  |  |  I_i^cur = I_i^new;
13 |  |  end
14 |  end
15 until convergence[1:N]=true ;
16 Return I;
```

Algorithm 2. The *CompImp* Algorithm

of each service operation to $1/N$ and then iteratively applying equation 2 until the importance values converge, i.e., for each operation, the difference between the old and the new importance value is less than some threshold c (typically, we can choose $c = 0.1\%$). The details of *CompImp* are shown in Algorithm 2.

Once the importance of all available web-service operations has been obtained, we simply define the importance of a web service as the average of the importance values of all operations in the service. The process is straightforward and not presented due to space limitations.

6.3 Combining Service Relevance with Importance

Recall that in section 4, each web service s is assigned a relevance score by similarity measure. In order to reflect the two factors we proposed for characterizing the user's preference degree for s, we need to incorporate a service importance score into the relevance score of s. Then, we can rank s according to its combination score, which is a weighted sum of its relevance score with a query q and its importance score. Formally, we have

$$Ranking\_Score(q, s) = \begin{cases} w \times R_s + (1 - w) \times I_s & \text{if } R_s > 0 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where $0 \leq w \leq 1$. Both R_s and I_s need to be normalized to between [0, 1]. A higher rank score indicates a more desirable web service, so the user's top-k search requirement can be satisfied.

7 Experiments and Evaluations

We have implemented a prototype system to evaluate the techniques presented in this paper. First, we investigate the time saving issue and present a service index structure that is used in our experiments; second, we evaluate the performance of building the service index; finally, we evaluate the effectiveness and efficiency of our IR-style search strategy.

The experiments were conducted on a P4 Windows machine with a 2GHz Pentium IV and 512M main memory. The data set used in our tests is a web service repository collected from [22,23,24]. Their WSDL specifications are available so we can obtain the textual descriptions and XML schemas of input/output data types. The data contains 223 web services including 930 web-service operations.

In order to improve the performance of searching for desired web services, we design a service index for the service repository. The service index keeps TF/IDF information about each WSDL document. Considering schema tree matching is time consuming, we also included an importance entry in the service index.

We first evaluated the efficiency of building the service index. The time performance of our algorithm is tested with the increase of the number of web-service operations. Taking CPU time as the standard measure, we get time costs of building the service index in Fig. 4(a), in which the time includes the costs of constructing relevance entry for all services, and the computation of the importance entry as well. Fig. 4(a) shows that, as the number of operations increases, the time cost of building the service index increases rapidly. This indicates that, by adding service operations, the number of schema trees increases significantly, leading to more cost of schema tree matching. However, the efficiency is still good since we take a fast tree edit distance method from [25].

We then evaluated the efficiency of searching for desired web services. The time cost is given in Fig. 4(b). It is can be seen that the time increases almost in a linear way with respect to the number of web service operations. This demonstrates that by building the service index, our searching performance is rather high, although building index is a bit time costing. But considering the fact that the web service repository does not change frequently comparing with a user's query request, the service index is effective and practical.

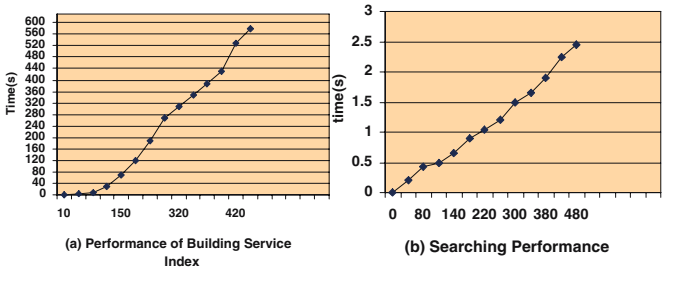

Fig. 4. Performance

8 Conclusions

In this paper, we have presented a novel IR-Style mechanism for discovering and ranking web services automatically, given a textual description of desired services. We have introduced the notion of preference degree for a web service, and suggested *relevance* and *importance* as two desired properties for measuring its preference degree. Also, various algorithms are given to obtain service relevance and importance. The key part for computing service importance is our schema tree matching algorithm, which catches not only structures, but even better semantic information of schemas defined in web services. Experimental results show the proposed IR-style search strategy is efficient and practical.

As part of on-going work, we are interested in improving efficiency of the connectivity computation algorithm in terms of running time, since the computation of extended tree edit distance is costly.

References

1. Booth, D., Haas, H., McCab, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture (2004), `http://www.w3.org/TR/ws-arch/`
2. Sajjanhar, A., Hou, J., Zhang, Y.: Algorithm for Web Services Matching. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 665–670. Springer, Heidelberg (2004)
3. Wang, Y., Stroulia, E.: Flexible Interface Matching for Web-Service Discovery. In: Proceedings of International Conference on Web Information Systems Engineering (WISE) (2003)
4. Dong, X., Halevy, A.Y., Madhavan, J., Nemes, E., Zhang, J.: Simlarity Search for Web Services. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 372–383 (2004)
5. Sycara, K.P., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. Autonomous Agents and Multi-Agent Systems 5(2), 173–203 (2002)
6. Roman, D., Lausen, H., Keller, U.: Web Service Modeling Ontology (WSMO). WSMO Final Draft 10 (2005)
7. Shen, Z., Su, J.: Web service discovery based on behavior signatures. In: Proceedings of International Conference on Services Computing (SCC), vol. 1, pp. 279–286 (2005)
8. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient IR-Style Keyword Search over Relational Databases. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 850–861 (2003)
9. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword Search in Relational Databases. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 670–681 (2002)
10. Agrawal, S., Chaudhuri, S., Das, G.: DBXplorer: A System for Keyword-Based Search over Relational Databases. In: Proceedings of International Conference on Data Engineering (ICDE) (2002)
11. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks 30(1-7), 107–117 (1998)

12. Pu, K., Hristidis, V., Koudas, N.: Syntactic Rule Based Approach to Web Service Composition. In: Proceedings of International Conference on Data Engineering (ICDE), p. 31 (2006)
13. Yu, C., Jagadish, H.V.: Schema summarization. In: VLDB, pp. 319–330 (2006)
14. Hao, Y., Zhang, Y., Cao, J.: WSXplorer: Searching for desired web services. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 173–187. Springer, Heidelberg (2007)
15. He, Q., Yan, J., Yang, Y., Kowalczyk, R., Jin, H.: Chord4s: A p2p-based decentralised service discovery approach. In: IEEE SCC (1), pp. 221–228 (2008)
16. Al-Masri, E., Mahmoud, Q.H.: Qos-based discovery and ranking of web services. In: ICCCN, pp. 529–534 (2007)
17. Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: WWW, pp. 795–804 (2008)
18. Salton, G., Wong, A., Yang, C.S.: A Vector Space Model for Automatic Indexing. Communications of the ACM (CACM) 18(11), 613–620 (1975)
19. Reis, D.D.C., Golgher, P.B., Silva, A.S.d., Laender, A.H.F.: Automatic web news extraction using tree edit distance. In: Proceedings of WWW Conference, pp. 502–511 (2004)
20. Zhang, K., Shasha, D.: Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. SIAM Journal on Computing 18(6), 1245–1262 (1989)
21. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley, New York (1990)
22. `http://www.xmethods.org` (XMethod)
23. `http://www.bindingpoint.com` (BindingPoint)
24. `http://www.webservicelist.com` (WebServiceList)
25. Nierman, A., Jagadish, H.V.: Evaluating structural similarity in xml documents. In: WebDB, 61–66 (2002)