An Intelligent Algorithm for Automatic Candidate Selection for Web Service Composition

Ashish Kedia, Ajith Pandel, Adarsh Mohata, and Sowmya Kamath S

Department of Information Technology,
National Institute of Technology Karnataka, Surathkal, Mangalore, INDIA
{ashish1294,ajithpandel,amohta163}@gmail.com
sowmyakamath@nitk.ac.in

Abstract. Web services have become an important enabling paradigm for distributed computing. Some deterrents to the continued popularity of the Web service technology currently are the non-availability of large-scale, semantically enhanced service descriptions and limited use of semantics in service life-cycle tasks like discovery, selection and composition. In this paper, we outline an intelligent semantics based web service discovery and selection technique that uses interfaces and text description of services to capture their functional semantics. We also propose a service composition mechanism that automatically performs candidate selection using the service functional semantics, when one Web service does not suffice. These techniques can aid application designers in the process of service based application development that uses multiple web services for its intended functionality. We present experimental and theoretical evaluation of the proposed method.

Keywords: Web Services Composition, Semantic Search, NLP

1 Introduction

W3C defines a Web Service as a software system that supports interoperable machine to machine interaction over a network. A web service can be uniquely identified by a URI and each is described using one or more XML based documents which define the service interfaces, the functionality provided by it and also prescribe the manner in which the it interacts with other systems. Large scale applications can be easily built by composing loosely coupled web services [10], thus enabling a service oriented architecture.

While developing applications, searching for an appropriate web service that can provide a required functionality is not trivial. Web Service discovery and retrieval often becomes a bottleneck. Recent years have not only seen an explosive increase in the number of web services being offered but have also witnessed a rise in number of standards to describe those service. The problem is further compounded by the fact that there is no central repository with all service descriptions. Web Service standards such as UDDI (Universal Description, Discovery and Integration) which relied on a central registry of all web services is now obsolete owing to its low benefit/complexity ratio. The requirements have also escalated. Developers now need a method to dynamically look-up for appropriate web service during run-time making service discovery a challenging task. Semantic Web Technology attempts to automate the web service discovery. Most of the existing algorithms for automated web service discovery serves to only web services that have explicit semantic tags associated with their description document which is an unreasonable expectation. A large number of existing web services do not have any semantics tags associated with their description document. Approaches to convert existing non-semantic description documents of web services to corresponding semantic ones are also severely limited.

Our work is focused on studying the existing methods of discovering web services and develop a method to automatically index a set of web services using their description documents such that services can be automatically searched and composed based on user's need. The rest of this paper in organized as follows. In section 2 we will discuss the existing work concerning the described problem. This is followed by section 3 that describes methodologies to index web services using their description documents. Section 4 talks about the algorithm to search the indexed web services to find services relevant to the user. In section 5 we propose a methodology to automatically compose multiple web services. In section 6 we will discuss the results obtained and analyze the proposed method. Finally, Section 7 concludes our work with a few possible future improvements.

2 Related Work

Yanbin et al [7] have modeled the service discovery problem as an assignment problem using functional constraints. They have proposed an automatic semantic search algorithm which is loosely based in assignment algorithm. It uses 3 step match making - Service Library Matchmaking, Service Matchmaking and Operation Matchmaking. Operation Matchmaking can be further divided into Interface Matchmaking and Concept Matchmaking.

Platzer and Dustdar [8] proposed the construction of a Vector Space to index descriptions of already existing services. They have used the prevalent information retrieval methods over the existing standards to create a multidimensional "term space", where each dimension represents a category of web services and then represent each web service in this space using a vector. The relative position of these vectors in the said space is used to compute the effective similarity of the corresponding web services.

Cuzzocrea et al [1] have considered both internal structure and component of web services. They have outlined an algorithm for service discovery that represents composite OWL-S (Web Ontology Language for Services) documents using graphs. They proposed an algorithm that matches a group of services with a query using such graph-based representation. They have not only considered the similarity of individual services in the matched group but have also taken into account the flow or control between different services of that group.

Sangers et al [9] have used popular NLP techniques like lemmatization, tagging parts of speech and word sense disambiguation to establish the semantics of web service description. They also determine the senses of the relevant words in user's query and then carry out a match-matching process between users query and indexed web services. In this method a context aware search is performed i.e., actual users need is matched with services that performs required computation. Fethallah et al [2] have outlined a mechanism to use the external interface (inputs/outputs) of web services. They have used domain ontology to classify service interfaces and then index the corresponding service conceptually. Once the services are indexed they use the popular co-sine similarity measurement to computer the degree of similarity between the query and the indexed services. The method yielded good result and is relative less resource intensive than the other existing methods.

Vector space search engine seems like a promising approach however it fails to account for the service semantics which is an important parameter in service discovery. Our algorithm tries to establish service semantics using text description and uses it as an additional parameter for service discovery over traditional vector space search to get the best of both methods. We also focus on serving user's need by automatically composing services whenever required.

3 Proposed Methodology

Figure 1 depicts the overall methodology adopted for the proposed system. We discuss each of these processes in detail below:

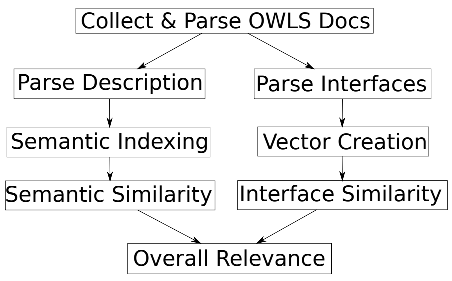

Fig. 1. Proposed Methodology

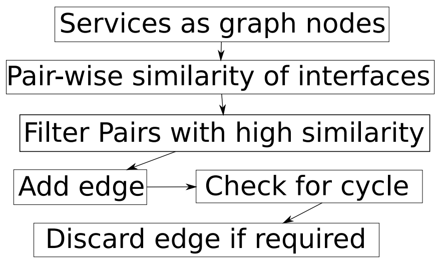

Fig. 2. Creation of Service Interface Graph

3.1 Preprocessing Web Service Description Documents

We propose using a combination of two methods for indexing web service description documents. The first method relies on the exact keywords that defines the service interface i.e., input and output. The second method relies on natural language processing techniques to derive the actual functionality provided by the service.

The OWL-S test collection[1] was used as a dataset for performing the experiments. We divided the services into seven popular categories like Economy, Education, Communication, Food, Travel, Weapon and Medical. It is difficult to categorize a vast of number of services into these categories strictly and thus we consider a vector space with 7 dimensions each representing a category of web services as listed earlier. Each category has a list of keywords associated with it which is denoted by C_i where $1 <= i <= 7$. We also allow a single keyword to be associated with more than one category.

3.2 Indexing Web Service Description Documents

Web service description files typically have tags like <profile:hasInput> and <profile:hasOutput> which specify the respective service interface. We use these interfaces to index the web services. To index a web service, we parse the description document associated with the service to extract the <profile:hasInput> and <profile:hasOutput>. After extracting we extract all the keywords used to describe both the service interface. Let us denote the list of keywords as W_i and W_o. We define two vectors namely V_i (Service Input Vector) and V_o (Service Output Vector). Both the vectors have seven elements (each representing a category) where each element is the number of keywords common to both the category and the service input (for V_i) / output (for V_o). Thus for each service two vectors are created and stored. Mathematically this can be formulated as shown in Equation 1 and 2.

$$V_{i_k} = \ \mid W_i \cap C_k \mid \ \forall \{k \mid 1 <= k <= 7\} \tag{1}$$

$$V_{o_k} = \ \mid W_o \cap C_k \mid \ \forall \{k \mid 1 <= k <= 7\} \tag{2}$$

3.3 Measuring similarity between services

The overall functionality of a web-service is typically described in a description document associated with web services using human-readable natural language. They tend to give more insight about the actual functionality provided by the web service. Natural language processing techniques can be used to extract the the real functionality provided by the web service. We have used a simple word sense disambiguation algorithm to choose the right meaning of all the words in the text description and then establish a context of the web service which can be used for matching the service with users requirements. A context is a set of meanings that represents the functionality provided by a web service.

The algorithm first extracts the text description and then eliminates frequently occurring words such as conjunctions, prepositions, etc, as they do not

[1] Available online at `http://projects.semwebcentral.org/projects/owls-tc/`

provide any significant information. Domain modeling can also be used to drop frequently occurring words. After this, all possible synsets of each word are obtained from Wordnet [5]. A context is initialized with all the disambiguated words i.e., words with single meaning. If no such words are found then a context is established by choosing the most frequently used meaning of a few words having relatively smaller number of synset. The algorithm simply chooses the meaning which is conceptually most similar to the already established context. To compute the similarity between a synset and a context we compute the average similarity of the given synset with each synset in the context as formulated by Equation 3.

$$sim(syn, context) = \overline{jcn(syn, s_i)} \ \forall \ s_i \ \epsilon \ context \tag{3}$$

where, *syn* is a synset of the given word. To find similarity between synsets we used the JCN Similarity algorithm [4]. After computing similarity of all the synset of a given word with the already established context, we choose the synset with maximum similarity and added it to the context. As such the final context has the set of most similar lemmas. We indexed each web service using the previously established context.

4 Searching Candidate Web Services

With every query user specify the required input, output and functionality. Based on the two methods of indexing, two different corresponding search methods can be used - Vector Space based Search and Semantic Search.

In the proposed system, the desired input and output specified by the user is used to construct two search vectors namely S_i and S_o representing the desired interface in the seven dimensional vector space described earlier. The process of converting the desired interface to corresponding vector is similar to that of converting service interfaces to corresponding vectors. Once we have the search vectors, we search for service vectors similar to search vectors. The similarity between two given vectors is determined using the cosine similarity score, described in Equation 4.

$$sim(V_1, V_2) = \frac{V_1.V_2}{||V_1|| * ||V_2||} \tag{4}$$

where, V_1 and V_2 are two vectors of same dimension. To find the total similarity between a search query and a service we compute the average similarity of input and output vectors respectively as illustrated in Equation 5.

$$\text{Total Similarity} = \frac{sim(R_i, S_i) + sim(R_o, S_o)}{2} \tag{5}$$

where, R_i and R_o are the input and output vector of an indexed service. We compute similarity with all the indexed services and then sort the result according to total similarity. After sorting we assign rank to each service.

The semantic search proceeds by iterating over the indexed services and selecting the services with similar context. A context of the user's query is established using the same method as described in previous section. The similarity between two context is computed using the Equation 6.

$$\text{Semantic Similarity} = \frac{\sum sim(u_i, v_i)}{m \times n} \tag{6}$$

where, u_i ϵ User's Query Context $\forall$ i = 0, 1, . . . n, v_i ϵ Service's Indexed Context $\forall$ i = 0, 1, . . . m and $sim()$ denotes similarity between 2 given lemmas. The services are sorted according to the context similarity score and each service is assigned a rank. The final rank of a service is computed as the average of rank assigned by each method. We give equal weightage to both the algorithms to compute the final result. However, the weight of each algorithm can be tuned according to the specific requirement.

5 Automatic Service Composition

It is often the case that a single service in the database is unable to satisfy user's query complete. In such cases we need to find multiple services that can work together in a given sequence so as to provide the required functionality to the user. In essence the services have to be automatically composed. Service composition is performed as follows:

- Searching for suitable web services that can be composed together to act as a single service
- Arranging the different web services in a particular sequence that yields the desired output
- Conversion of data-formats so that output of one service matches the input format expected by the next service in the sequence

Several solutions to this service composition problem have been proposed based on graphical model of web services [6], [3]. In this section, a methodology to search for multiple web services that can be composed together to serve the user's need, using a graph of interconnected web services is discussed.

5.1 Constructing a Service Interface Graph

A DAG (Directed Acyclic Graph) is constructed to model services and the relation between their interfaces. Each node in this graph represents a web service. A node has several incoming and outgoing edges. An edge from node 'A' to node 'B' signifies that the output yielded by service 'A' is similar to the input accepted by service 'B' i.e., service 'A' and 'B' can be composed together. To construct the graph we first compute the equivalent input and output vector of all the service in the database. Then, we match the output of each service to the input of every other service i.e., determine the co-sine similarity between the output vector of first service and input vector of second service. If the similarity is found to be greater than a pre-determined cut-off then the two services are connected via an edge from first one to the second. After the addition of each edge, the graph is checked for cycles. If any cycles are found to exist in the graph, the newly added edge is discarded. Figure 2 illustrates a flow chart showing all the steps involved in creation of the said graph.

The main objective of this process is, to model Web services such that the service composition problem can be treated as a simple graph traversal problem. Thus, it is essential to have an acyclic graph. The similarity cut-off for adding edges is chosen to be 0.9. A high value is chosen to ensure that there is almost a perfect match between the interfaces. This cutoff can be determined dynamically based on the average similarity of interfaces and several other domain-dependent factors.

5.2 Executing a Service Composition Query

Once the graph is constructed, it has to be traversed for each query. Firstly, an input and output vector corresponding to the user's query is computed. Then a keyword based query as described in previous section is executed. The resultant services are sorted according to the input similarity - and top results (top k) are filtered. This gives the top k start-points of potential composition. For each of these input services, the graph is traversed using the well known DFS (Depth First Search) Algorithm starting from the input service as source. For every node visited, the similarity between the node's output vector and user's query output vector is determined. Among all the nodes visited, the node with best output vector is selected. The path between the corresponding source node and the node with best output yields the best composition possible for the corresponding source service.

6 Experimental Results and Analysis

The results that we obtain are very encouraging. We are able to obtain very relevant web services given an interface. We have obtained this results with a very small set of keywords in each field (average 40 each). As such, with a large set of keywords in each category we should be able obtain much better results. A few sample Query and their corresponding results have been listed in Table 1. The cosine similarity values obtained for each service in the result set is also mentioned.

In this section, an estimate of the asymptotic run-time complexity analysis of all the major steps involved in our proposed algorithm is presented. The first step is to parse the description document of a service which is dependent on the length of the description document. Since the length of the description document is roughly same for all services, this step takes constant time. To construct the vectors that can represent the service in 7-D vector space we have to search through all the keywords belonging to all the categories. Thus

Table 1. Observed results for some sample queries

Input	Output	Services and Similarity
Car	Price	3wheeledcar_price_service (1.0), car_price_service (1.0), citycity_arrowfigure_service (1.0), lenthu_rentcar_service (0.972)
Missile	Range	missile_lendingrange_service(0.971), missile_givingrange(0.933), ballistic_range_service(0.918)
Location	Distance	sightseeing_service(1.0), DistanceInMiles(0.908), calculate_betwee_Location(0.903), surfing_service(0.901)
Medical	Bed	hospital_investigatingaddress_service(0.789), medicalclinic_service(0.670), SeePatientMedicalRecords_service(0.640)

constructing vectors take $\mathcal{O}(C)$ time, where C is the number of keywords across all categories. Thus the complexity of indexing N services is $\mathcal{O}(NC)$. The next step is construction of DAG. The input vector of each service is matched with output vector of every other service. Thus the complexity of graph construction is $\mathcal{O}(N^2)$, where N is the number of service. The check for cycle formation is linearly dependent on the number of nodes in graph and thus it doesn't add anything to the complexity. However, the graph is constructed only once when the server is started and thus we can afford it to be slower. Whenever a new service is added to the database i.e., A new node is appended to the graph, it's input and output has to be matched with every other service and thus the complexity of adding new node will be $\mathcal{O}(N)$.

The next step is executing user's query. The complexity of constructing query vector is again $\mathcal{O}(C)$. Finding Co-Sine similarity between query vector and a service takes constant time. Thus the overall complexity of vector space search is $\mathcal{O}(N + C)$. Constructing query context will also take constant time since the number of keywords given by user as a part of their query will typically have a constant upper-bound. Again the time complexity of traversing the whole data-set and compute context similarity is $\mathcal{O}(N)$. Once we filter top K result we have to rank them which takes $\mathcal{O}(K \log K)$ time. In a practice, each node will have very few outgoing edges on an average. Thus, assuming the number of edges in the graph is proportional to the number of nodes, the time complexity to execute a service composition query is $\mathcal{O}(N)$.

7 Conclusion and Future Work

In this paper, a novel approach for automatically determining service composition candidates for a given user requirement is presented. The proposed method offers a lot of scope for further improvements. Firstly, we have manually labeled each category with corresponding keywords but the system should be able to learn keywords for each category automatically over time using Machine Learning techniques. Secondly, the weightage to results obtained from multiple approaches can be tuned for different domains to obtain better results. Thirdly, domain modeling concepts can be used to improve the word sense disambiguation of the synsets obtained. Certain synsets that have no relevance in a given domain can easily be eliminated by this approach. As the information content of different words is also domain dependent, words frequently encountered in a given domain can be discarded while parsing the description documents. This can help in further optimization and improvement in the performance of the proposed methodology during the process of determining service composition candidates.

References

1. Cuzzocrea, A., Fisichella, M.: Discovering semantic web services via advanced graph-based matching. In: Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on. pp. 608–615. IEEE (2011)
2. Fethallah, H., Chikh, A.: Automated retrieval of semantic web services: a matching based on conceptual indexation. Int. Arab J. Inf. Technol. 10(1), 61–66 (2013)
3. Hashemian, S., Mavaddat, F.: A graph-based approach to web services composition. In: Applications and the Internet, 2005. Proceedings. The 2005 Symposium on. pp. 183–189 (Jan 2005)
4. Jiang, J.J., Conrath, D.W.: Semantic similarity based on corpus statistics and lexical taxonomy. CoRR cmp-lg/9709008 (1997), http://arxiv.org/abs/cmp-lg/9709008
5. Miller, G.A.: Wordnet: a lexical database for english. Communications of the ACM 38(11), 39–41 (1995)
6. Oh, S.C., On, B.W., Larson, E., Lee, D.: Bf*: Web services discovery and composition as graph search problem. In: e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on. pp. 784–786 (March 2005)
7. Peng, Y., Wu, C.: Automatic semantic web service discovery based on assignment algorithm. In: 2010 2nd International Conference on Computer Engineering and Technology. vol. 6 (2010)
8. Platzer, C., Dustdar, S.: A vector space search engine for web services. Third European Conference on Web Services (2005)
9. Sangers, J., Frasincar, F., Hogenboom, F., Chepegin, V.: Semantic web service discovery using natural language processing techniques. Expert Systems with Applications 40(11), 4660 – 4671 (2013), http://www.sciencedirect.com/science/article/pii/S0957417413001279
10. Truong, H.L., Dustdar, S.: A survey on context aware web service systems. International Journal of Web Information Systems 5(1), 5–31 (2009)