Graph-Based Matching of Composite OWL-S Services

Alfredo Cuzzocrea[1], Juri Luca De Coi[2], Marco Fisichella[2], and Dimitrios Skoutas[2]

[1] ICAR-CNR and University of Calabria, Italy
cuzzocrea@si.deis.unical.it

[2] Forschungszentrum L3S, Hannover 30167, Germany
{decoi,fisichella,skoutas}@L3S.de

Abstract. Existing techniques for Web service discovery focus mainly on matching functional parameters of atomic services, such as inputs and outputs. However, one of the main advantages of Web services is that they are often composed into more complex processes to achieve a given goal. Applying such techniques in these cases, ignores the workflow structure of the composite process, and therefore may produce matches that are not very accurate. To overcome this limitation, we propose in this paper a graph-based method for matching composite services, that are semantically described as OWL-S processes. We propose a graph representation of composite OWL-S processes and we introduce a matching algorithm that performs comparisons not only at the level of individual components but also at the structural level, taking into consideration the control flow among the atomic components. We also report our preliminary results of our experimental evaluation.

1 Introduction

Web services are a key technology for enabling interoperability and software reuse. Service discovery is the process of matching a service request with a service advertisement, and it is based on comparing their descriptions, such as their input and output parameters. Service composition deals with composing services to create complex processes that achieve a desired goal given an initial state. This is an important feature, since it allows atomic services to be combined in a flexible way to complete complex tasks. In the Semantic Web, service descriptions are semantically annotated using concepts from domain ontologies in order to facilitate and improve the precision of their automatic discovery and composition.

Service composition is a very challenging task, either when performed at design time or, especially, online. Given also that reusability is a key concern in service-oriented architectures, this makes the discovery of existing composite services an important problem. When an application needs to create a composite service to fulfill a given goal, it is more effective and efficient to first search a repository of existing compositions to find similar ones. Then, the best matches identified can be modified, extended or combined, to produce the desired composite service instead of composing one from scratch. Moreover, when browsing a repository of composite processes, the user may find some interesting process and then issue a "more like this" query to retrieve additional results.

In this paper, we focus on composite Semantic Web services described in OWL-S [3], since OWL-S provides different parts for explicitly describing the *profile* and the *model* of a service. The service profile is mainly aimed at supporting service discovery, and it includes the functional parameters of the service, which are the ones typically used for the matchmaking between service descriptions. The service model is primarily aimed at the specification of composite services. In particular, it describes the internal components and the control flow of the composite process. In a typical service discovery scenario, a query is formulated as the description of a desired service and the result is a ranked list of advertised services, the description of which matches the request, according to a matchmaking algorithm that employs one or more matching criteria. Hence, existing discovery methods do not differentiate between atomic and composite services. The service profile is also used for the matchmaking of composite services, which means that a complex process is treated as a "black box"; its most integral part, the process model, is not taken into account. This severely reduces the accuracy of the results, introducing both false positives and false negatives.

To address this problem, we propose a graph-based method for matching composite services. Matchmaking is performed on the service model rather than the service profile, which includes the structural part of the composite service. In OWL-S, composite services can be composed from atomic ones or from other simpler composite services, allowing several levels of nesting. To specify how component services are combined together, a set of *control constructs* is provided, similar to the typical control structures found in programming languages. These allow services to be executed sequentially, in parallel, conditionally or in a loop.

The proposed method performs the matching on two levels. It matches both the atomic services of the composite process, using their service profiles, as well as the way these services have been composed to create the composite process. To avoid the details and specificities of the OWL-S process model, the composite process is first transformed to a graph representation, containing its component services and their interactions. Matchmaking is then performed considering node similarities and finding common (sub)structures between the two graphs that represent the requested and the available composite process. To increase the efficiency of the search, a two step approach is followed. Initially, a set of candidate graphs is identified considering mappings between pairs of nodes. Then, the best candidates are selected and their structural similarity to the query graph is taken into account in order to filter out false positives and determine the final ranking of the results.

The rest of the paper is structured as follows. The next section describes our graph-based representation of composite OWL-S processes. Section 3 introduces the matchmaking algorithm. A preliminary experimental evaluation is presented in Section 4. Section 5 discusses related work, and Section 6 concludes the paper with directions for future work.

2 Graph Representation of OWL-S Processes

The OWL-S description of a Web service comprises three main parts. The *Service Profile* specifies the functional parameters of the service, namely inputs, outputs, pre-conditions and effects; it may also contain information about the provider and the category of the service, as well as plain text description. The *Service Model* describes the components and the structure of a composite process. The *Service Grounding* specifies the details required by an agent to invoke the service, such as communication protocol, message formats and port numbers. In contrast to typical service discovery approaches that rely on the service profile for matching atomic services, our matchmaking algorithm utilizes the information provided by the service model to perform graph-based matchmaking between complex processes.

In our approach, we represent composite processes as graphs in order to facilitate their matching. Given a composite OWL-S process P, we describe below how the corresponding graph representation, denoted as G_P, is derived. Let $\mathcal{C}$ be the set of control constructs supported in OWL-S. Each occurrence of a control construct $C \in \mathcal{C}$ is represented by a pair of nodes, C_b and C_e, that denote its begin and its end part, respectively. We denote the sets of such nodes as $\mathcal{C}_b$ and $\mathcal{C}_e$, respectively. These nodes allow us to represent the part of the process that is enclosed by this control construct and, hence, to represent the nesting of processes. Moreover, each atomic service s in a composite process is represented by a graph node s. For the sake of simplicity, we use the same symbol to refer both to the node and to the service it represents, since the distinction is typically clear from the context. The set of all the atomic services that are contained in the composite process P is denoted as $\mathcal{S}_P$. Thus, a composite process P is represented by a graph $G_P = (V, E)$, with node set $V = \mathcal{C}_b \cup \mathcal{C}_e \cup \mathcal{S}_P$ and edge set $E \subseteq (\mathcal{C}_b \times \mathcal{C}_b) \cup (\mathcal{C}_b \times \mathcal{S}_P) \cup (\mathcal{S}_P \times \mathcal{S}_P) \cup (\mathcal{S}_P \times \mathcal{C}_e) \cup (\mathcal{C}_e \times \mathcal{C}_e)$. The edges in the graph denote the control flow, as it will be explained below. In the following, we list the control constructs provided by OWL-S.

– *Sequence (SQ).* It encloses a list of components to be executed in the specified order.
– *AnyOrder (AO).* It encloses a bag (according to the OWL-S definition) of components to be executed sequentially, but without imposing any restriction on the ordering.

Table 1. Construction of the process graph

Control construct	*Added edges*	Control construct	*Added edges*
`Sequence`$(s_1,\ s_2,\ \ldots s_{n-1},\ s_n)$	(SQ_b, s_1), (s_1, s_2), $\ldots$, $\ldots$, (s_{n-1}, s_n), (s_n, SQ_e)	`AnyOrder`$(s_1,\ s_2,\ \ldots s_{n-1},\ s_n)$	(AO_b, s_1), (s_1, s_2), $\ldots$, $\ldots$, (s_{n-1}, s_n), (s_n, AO_e)
`Split`$(s_1,\ s_2,\ \ldots s_n)$	(SP_b, s_1), (SP_b, s_2), $\ldots$, (SP_b, s_n), (s_1, SP_e), (s_2, SP_e), $\ldots$, (s_n, SP_e)	`SplitJoin`$(s_1,\ s_2,\ \ldots s_n)$	(SJ_b, s_1), (SJ_b, s_2), $\ldots$, (SJ_b, s_n), (s_1, SJ_e), (s_2, SJ_e), $\ldots$, (s_n, SJ_e)
`Repeat` (s) `While` $(cond)$	(RW_b, s), (s, RW_e)	`Repeat` (s) `Until` $(cond)$	(RU_b, s), (s, RU_e)
`Choice`$(s_1,\ s_2,\ \ldots s_n)$	(CH_b, s_1), (CH_b, s_2), $\ldots$, (CH_b, s_n), (s_1, CH_e), (s_2, CH_e), $\ldots$, (s_n, CH_e)	`If` $(cond)$ `Then` (s_1) `Else` (s_2)	(IF_b, s_1), (IF_b, s_2), (s_1, IF_e), (s_2, IF_e)

- *Split (SP).* It encloses a bag of components to be executed in parallel.
- *SplitJoin (SJ).* It encloses a bag of components to be executed in parallel. The difference between `SP` and `SJ` is that the latter specifies barrier synchronization, i.e., all the included components need to finish their execution before the control construct is considered to be finished.
- *Choice (CH).* It encloses a bag of components, one of which can be chosen for execution.
- *IfThenElse (IF).* It encloses two components, one of which is executed based on whether a specified condition is true or false.
- *RepeatWhile (RW).* It encloses a component that is executed in a loop, as long as a specified condition is true.
- *RepeatUntil (RU).* It encloses a component that is executed in a loop, until a specified condition becomes true.

Note that we do not include conditions in our graph representation and in our matching algorithm. This is out of scope of this paper, given that OWL-S does not dictate any specific language for expressing such logical conditions. A possible extension to address the issue of conditions is to include them as labels on the nodes that correspond to control constructs having a condition, or on the outgoing edges of these nodes. Then, during the matching, an appropriate reasoner for the language used to express these conditions can be invoked to determine the degree of similarity between the condition in the requested service and the one in the advertised service, e.g., by inferring whether one condition implies the other (or its negation).

Table 1 specifies how the graph edges are constructed for each of the OWL-S control constructs. For simplicity, the table assumes only atomic services as components inside a control construct. If instead of an atomic service s there exists a nested composite process P' enclosed by a control construct C, then: (a) an edge (v, C_b) is added instead of each incoming edge (v, s); (b) an edge (C_e, v) is added instead of each outgoing edge (s, v); (c) the representation of the subprocess P' is computed recursively and added to the graph. Note that for each new occurrence of a control construct, a new pair of corresponding begin and end nodes is introduced. Some examples are shown in Figure 1.

3 Matching OWL-S Processes

In this section, we present our matching algorithm for composite OWL-S processes. First, we discuss how the degree of match *dom* is computed between atomic components and then how the structural similarity is taken into account.

3.1 Matching Atomic Components

Let G_R and G_P be the graph representations of a requested and a candidate composite services R and P, respectively. In the following, we show how to compute the degree of match between two nodes $r \in V(G_R)$ and $s \in V(G_P)$. Recall from Section 2 that each node in the graph corresponds either to an atomic service or to the begin or end part of a control construct. We compute the degree of match only between nodes of the same type, i.e., only for the cases that: (a) $r \in \mathcal{S}_R$ and $s \in \mathcal{S}_P$; or (b) $r \in \mathcal{C}_b$ and $s \in \mathcal{C}_b$; or (c) $r \in \mathcal{C}_e$ and $s \in \mathcal{C}_e$. For all other combinations, the degree of match is zero.

First, we define the degree of match between nodes that correspond to atomic services (also denoted by r and s). The degree of match is computed based on the input and output parameters of these services. For atomic services, an offer s matches a request r if: (a) the outputs offered by s match the outputs requested by r, and (b) the inputs provided by r match the inputs required by s. Consequently, to compute the degree of match for the inputs of r and s, we find the best match for each input of s and we normalize based on the number of input parameters of s:

$$dom_{IN}(r,s) = \frac{\sum\limits_{u \in IN_s} \max\limits_{v \in IN_r} \{sim(u,v)\}}{|IN_s|} \tag{1}$$

Similarly, for the outputs of r and s, we find the best match for each output of r and we normalize based on the number of output parameters of r:

$$dom_{OUT}(r,s) = \frac{\sum\limits_{u \in OUT_r} \max\limits_{v \in OUT_s} \{sim(u,v)\}}{|OUT_r|} \tag{2}$$

In Equations 1 and 2, IN_p and OUT_p denote, respectively, the set of input and output parameters of an atomic process p. The function sim computes the similarity between two individual input or output parameters. There are two basic alternatives for defining this function. The first one is to compare the corresponding parameter classes u' and v' in the ontology $\mathcal{O}$, as defined in the OWL-S service descriptions. In this case, we compute the similarity based on the number of common ancestors of these two classes:

$$sim(u,v) = \frac{|\{w \in \mathcal{O} \mid u' \sqsubseteq w \wedge v' \sqsubseteq w\}|}{|\{w \in \mathcal{O} \mid u' \sqsubseteq w\} \cup \{w \in \mathcal{O} \mid v' \sqsubseteq w\}|} \tag{3}$$

The second alternative is to compare the parameter names using some common string similarity measure, such as cosine similarity or Jaccard similarity. A hybrid similarity measure, combining both alternatives, can also be used [11].

The overall degree of match between r and s is then computed by aggregating the partial degrees of match for the inputs and the outputs, e.g., as the (weighted) average. This can be extended to include scores derived from additional matching criteria.

Next, we discuss how to define the similarity between nodes corresponding to control constructs. As shown by the description of the OWL-S control constructs in Section 2, some of them have similar functionality, and therefore, replacing one with another when searching for similar composite processes should incur a lower penalty. We examine the following cases.

Sequence *and* AnyOrder. Both of these control constructs specify the execution of components in sequence; the difference between the two is that the latter does not specify the exact order but instead it allows any possible ordering (as long as there are no overlaps in the execution of two different components). Hence, a node SQ in the graph of the requested process R can be matched with a node AO in the graph of a candidate process P with a low effect on the similarity of the two processes. This case, however, is not symmetric. If AO appears in R and SQ in P, then the offered process is more restrictive than the requested one; hence, the match should be allowed, but with a lower score.

Split *and* SplitJoin. Both of these control constructs specify the execution of components in parallel; the difference is that the latter specifies also barrier synchronization. Hence, it is permitted to match one of them with the other one with low effect on the similarity of the two processes.

RepeatWhile *and* RepeatUntil. Both of these control constructs specify the execution of the enclosed component in a loop; their difference consists in whether the condition is checked at the beginning or at the end of each iteration. Therefore, matching these two control constructs should have a low effect on the process similarity.

Based on these observations, we define the similarity between two nodes denoting control constructs as follows:

$$dom_{\mathcal{C}}(r,s) = \begin{cases} 0.9 & \text{for the pair (SQ, AO)} \\ 0.5 & \text{for the pair (AO, SQ)} \\ 0.8 & \text{for the pairs (SP, SJ), (SJ, SP),} \\ & \text{(RW, RU) and (RU, RW)} \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

These values hold when both of the nodes correspond either to the begin or to the end part of a control construct; otherwise, the degree of match is zero.

3.2 Matching Process Structure

To take into account the workflow structure of composite services, we measure the “overlap” in their graph representations. For this purpose, we compute their maximum common subgraph. This technique is often used in other applications, such as searching and mining databases of chemical structures, pattern recognition or computer vision [2].

Given a graph $G = (V, E)$, a subgraph of G is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' = E \cap (V' \times V')$. Moreover, a graph $G = (V, E)$ is isomorphic to another graph $G' = (V', E')$ if there exists a bijective function $f : V \to V'$ such that for any edge $e = (v_1, v_2) \in E$ there exists an edge $e' = (f(v_1), f(v_2)) \in E'$. Then, the maximum common subgraph MCS of two graphs G_1 and G_2 is defined as the largest subgraph of G_1 that is isomorphic to a subgraph of G_2. Given the above, we can define the degree of match between a request graph G_R and a candidate graph G_P.

Definition 1. *Let G be the maximum common subgraph between a request graph G_R and a candidate graph G_P and f be the corresponding bijective function that maps G to a subgraph of G_P. The degree of match between G_R and G_P is defined as:*

$$dom_{\mathcal{G}}(G_R, G_P) = \frac{\sum\limits_{v \in V_G} dom_{\mathcal{V}}(v, f(v))}{|G_R|} \tag{5}$$

where V_G denotes the set of nodes of graph G, $|G_R|$ is the number of nodes in the query graph and $dom_{\mathcal{V}}$ is a function that computes the degree of match between pairs of graph nodes based on their type, as described in Section 3.1. Notice that the MCS of two graphs is not necessarily unique; in that case, the one with the highest degree of match is considered.

A problem that arises from this approach has to do with the computational complexity, since the maximum common subgraph isomorphism problem is known to be NP-hard. Therefore, reducing the number of maximum common subgraphs to be computed becomes a critical issue. We address this problem based on the following observation. From Equation 5, it can be seen that, in order for two graphs to have a large degree of match, they should have a large number of node pairs with high degree of match. Indeed, if there are only a few nodes in G_R that can be mapped with high similarity to nodes of G_P, then the sum in the numerator of Equation 5 can not be large. However, this is a necessary but not sufficient condition, since the sum is computed over the nodes of the identified maximum common subgraph. Hence, if the two graphs have many similar nodes but a small maximum common subgraph, then the sum would be again small. This is desired in order to prevent matches between composite services that are different from a structural point of view.

To make the search process more efficient, we identify first those candidate graphs that have nodes that can be mapped with high similarity to the nodes of the query graph, and we select the top-k' ones. This provides a list with candidate matches for the query which contains also false positives due to the reason explained previously. Then, we apply Equation 5 on this subset in order to compute the actual degree of match and to obtain the final list of top-k matches, after filtering out the false positives. The value of k' has to be larger than k to account for the presence of false positives, but it can still be significantly smaller than the total number of candidate graphs to be examined.

To obtain the top-k' list of candidate graph matches, we apply a process based on the Hungarian algorithm (also referred to as the Kuhn-Munkres algorithm) [13], which has also been applied in a similar way to provide an approximation for the graph edit distance [15]. The algorithm can be used to solve the assignment problem in polynomial time and relies on a square cost matrix $\{c^i_j\}$, where each element c^i_j represents the cost of assigning the job j to the worker i. The output of the algorithm is the assignment minimizing the overall cost. We use this to compute the optimal assignment between the nodes of the query graph G_R and those of the candidate graph G_P. Based on the similarity between graph nodes, computed as described in Section 3.1, we construct a $|G_R| \times |G_P|$ cost matrix, where the cost for each pair of nodes is calculated as $c_v^u = 1 - dom_{\mathcal{V}}(u, v)$. In the general case, the number of nodes of the two graphs is not the same, which means that not every node of the one graph can be mapped to a node of the other. To deal with this case, we introduce the concept of ε-node. The assignment of a node v to an ε-node (resp. of an ε-node to a node v) denotes that there is no mapping from (resp. to) node v. In other words, this corresponds to removing (resp. adding) a node in the graph. To make the cost matrix a square matrix, we introduce $||G_R| - |G_P||$ ε-nodes and we add the corresponding rows or columns in the matrix, as needed. We also set the cost for an assignment involving ε-nodes to 1. The optimal assignment between nodes is provided by the output of the algorithm. The overall cost of the assignment is computed as the sum of the costs of the pairwise mappings. The results are sorted in increasing order of cost and the top-k' ones are selected. For each one of these results, the degree of match to the query is then computed according to Equation 5, as explained previously, in order to obtain the final ranking.

4 Experimental Evaluation

Since existing approaches to service discovery focus on atomic services, we are not aware of an appropriate benchmark for evaluating the task of composite service matchmaking. To overcome this limitation, we have conducted experiments on a synthetically generated dataset of composite OWL-S processes, which were composed randomly from a set of publicly available real-world atomic OWL-S services. We describe first our experimental setup and methodology and then we present our results.

We implemented a synthetic generator for composite OWL-S processes. For each process, the generator first selects randomly one control construct, and then chooses how many atomic services or control constructs will be nested in it. This number is bound by a minimum and maximum value specified in a configuration file, which also defines the probability for selecting an atomic service or a given control construct. The number of maximum nested levels for control constructs is also specified. The atomic OWL-S services are selected from the OWLS-TC v2 collection[1], which is a publicly available collection of OWL-S services used to evaluate and compare different matchmaking algorithms. It comprises 1007 services from 7 different domains. All these are atomic services, hence we could not use the provided queries and their corresponding relevance sets to evaluate our matchmaking algorithm for composite processes. Using the generator, we created a dataset comprising the graph representations of 100 composite OWL-S processes and 10 queries.

We have implemented the matchmaking algorithm described in Section 3 in Java, reusing existing libraries whenever possible. In particular, we used the OWL-S API[2] for parsing the descriptions of OWL-S services in order to match

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

[2] `http://www.mindswap.org/2004/owl-s/api/`

their inputs and outputs, the SimPack[3] library for the computation of the maximum common subgraph and a Java implementation of the Hungarian algorithm[4].

For each one of the 10 queries, we first ran the matchmaking process based on the Hungarian algorithm to obtain a candidate list of matches, and we selected the 20 graphs with the lowest assignment cost. These are graphs that contain nodes with high similarity to the nodes of the query graph, but do not necessarily match the structure of the requested process. Then, for each one of these top-20 candidate matches, we computed the degree of match to the query based on the maximum common subgraph and we retrieved the top-10 results.

We compared the lists of top-10 graphs before ($\mathcal{L}_H$) and after performing the last step ($\mathcal{L}_{MCS}$). Our purpose was to examine how much the former ranking differs from the latter one, i.e., for how many graphs and how much the ranking changes once the structural similarity between the query and candidate processes is taken into account. For this purpose, we used Spearman's footrule distance [10], a commonly used distance measure for comparing different rankings. In particular, we used the extended version proposed by Fagin et al. [6], denoted in the following by F^*, which handles also the case where the compared rankings do not refer to the same set of items. This measure can be applied to our case as follows:

$$F^*(\mathcal{L}_H, \mathcal{L}_{MCS}) = \frac{\sum\limits_{i \in \mathcal{G}} |pos(i, \mathcal{L}_H) - pos(i, \mathcal{L}_{MCS})|}{maxF^*} \tag{6}$$

where $\mathcal{G}$ denotes the set of graphs in the two ranked lists and the function $pos(i, \mathcal{L})$ returns the position of i in the list $\mathcal{L}$ if $i \in \mathcal{L}$ and $|\mathcal{L}| + 1$ otherwise. $maxF^*$ denotes the maximum possible value that the *numerator* can take, which equals to $n(n+1)$, assuming that the lists to be compared consist of n elements. Higher values of this measure indicate higher difference between the rankings. For two identical rankings the value is zero, whereas the maximum value 1 is obtained when the two lists do not have any elements in common.

Fig. 2 shows the Spearman's distance between the $\mathcal{L}_H$ and $\mathcal{L}_{MCS}$ rankings for our 10 queries. As shown, in all cases the set of results and/or their ranking is affected after the structural similarity is taken into account. This is because some of the initial matches are identified as false positives and they are removed or ranked lower.

We examine in more detail how the results change for queries 1 and 3, which are the ones with the lowest and highest Spearman's distance, respectively. Since the query graphs are also contained in the dataset, the top-1 result in all cases is an exact match with the query graph itself. Table 3 shows the graph IDs in the $\mathcal{L}_H$ and $\mathcal{L}_{MCS}$ rankings for these two queries. Notice, for example, how graph 73, which does not appear in the $\mathcal{L}_H$ list of query 1, is ranked 4th in the $\mathcal{L}_{MCS}$ for the same query, whereas graph 42, initially at the 5th rank, does not appear

[3] `http://www.ifi.uzh.ch/ddis/simpack.html`

[4] `http://sites.google.com/site/garybaker/hungarian-algorithm/assignment`

Table 2. Spearman's distance for each one of the 10 queries

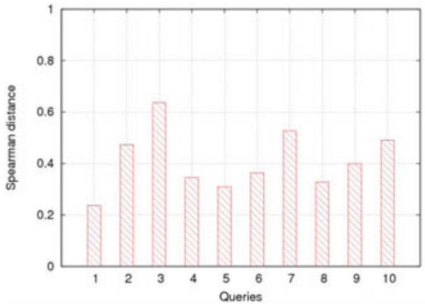
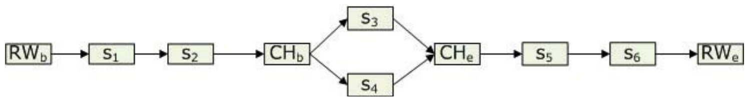
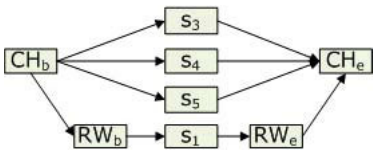


Table 3. Top-10 graphs for queries 1 and 3

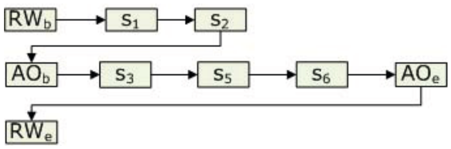
Query 1		Query 3	
$\mathcal{L}_H$	$\mathcal{L}_{MCS}$	$\mathcal{L}_H$	$\mathcal{L}_{MCS}$
9	9	27	27
10	62	3	62
62	10	83	73
6	73	71	96
42	6	96	57
22	22	28	99
89	35	95	9
35	46	73	35
30	89	76	53
3	83	53	10



(a) Query graph 1



(b) Rank 2 result in the $\mathcal{L}_H$ list



(c) Rank 2 result in the $\mathcal{L}_{MCS}$ list

Fig. 1. A sample of query results

in the final list of the top-10 results. The differences are even more apparent for the two result lists of query 3.

As an illustrative example, Fig. 1 shows the results at rank 2 returned by the algorithm taking (Fig. 1c) and not taking (Fig. 1b) structural similarity into account for query 1 (Fig. 1a). The result returned at rank 2 in the $\mathcal{L}_{MCS}$ list is clearly more similar to the query than the one returned in the $\mathcal{L}_H$ list. Structural similarity plays a major role in realizing that (c) is more similar to (a) than (b), despite the fact that (a) and (b) share a higher number of nodes, and for this reason result to a lower assignment cost for the Hungarian algorithm.

5 Related Work

Traditional service discovery approaches employ IR-based techniques, such as keyword search on the textual descriptions of the services or matching of parameter names using common string similarity measures. A clustering algorithm is used in [5] to group parameter names into semantically meaningful concepts, which are used to identify similar services. An online search engine for Web services is *seekda*[5], which crawls and indexes service descriptions from the Web. Users can search for services using keywords, tag cloud navigation or faceted browsing, e.g., by country or service provider.

For services on the Semantic Web, logic-based matching is applied to increase the accuracy of the discovery process [14,12]. A reasoner is used to infer equivalence, subsumption or disjointness between the ontology classes describing the compared service parameters and the type of match is characterized accordingly as *exact*, *plug-in*, *subsumes*, *subsumed-by* or *disjoint*. In [1], the problem of matching requested and offered parameters in Semantic Web service descriptions is modeled as the one of matching bipartite graphs. Furthermore, the degree of match can be computed as a continuous, normalized value in the $[0, 1]$ interval, by defining some similarity measure between classes in the ontology [4,18]. Hybrid solutions have also been proposed for combining IR and logic-based techniques [11,9]. Ranking match results combining multiple matching criteria has been proposed in [17].

However, all the aforementioned approaches deal with the discovery of atomic services, ignoring the internal structure and components of a composite process. Our approach addresses this limitation by proposing a graph-based matchmaking algorithm for composite services.

Further work has dealt with the problem of workflow discovery. A search engine for workflows has been presented in [16], which allows for keyword queries to be issued over workflows. A workflow is retrieved if it contains components that match the keywords in the query. *myExperiment*[6] is another search engine for scientific workflows. Again, search is based on keyword queries or tags. In [7], workflow descriptions are augmented with constraints derived from properties about the workflow components used to process data, as well as the data itself. However, structural similarity is also not taken into account during matchmaking. Finally, [8] presents an approach and a tool to discover workflows that employs matching at the workflow structure level. However, they consider generic workflows and therefore they do not deal with how to match individual components or how to handle control constructs. To the best of our knowledge, our method is the first one to address the problem of discovering composite OWL-S services.

6 Conclusions and Future Work

We have proposed a graph-based method for matching composite OWL-S services. In contrast to existing approaches, which deal with atomic services, we focus on the internal components and structure of composite services and we perform the matching based on their process model. We employ a graph representation of composite OWL-S services, where the nodes represent atomic services and OWL-S control constructs. Based on this, the matching algorithm computes the degree of

[5] `http://seekda.com/`

[6] `http://www.myexperiment.org/`

match between two composite processes based on both node similarity and structural similarity.

As future work, we plan to conduct a more thorough experimental evaluation, and to extend our algorithm to consider conditions on graph nodes, as well as graph indexing to increase search efficiency.

References

1. Bellur, U., Kulkarni, R.: Improved matchmaking algorithm for semantic web services based on bipartite graph matching. In: ICWS, pp. 86–93 (2007)
2. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters 19(3-4), 255–259 (1998)
3. Burstein, M., et al.: OWL-S: Semantic markup for web services. In: W3C Member Submission (November 2004)
4. Cardoso, J.: Discovering semantic web services with and without a common ontology commitment. In: IEEE SCW, pp. 183–190 (2006)
5. Dong, X., Halevy, A.Y., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: VLDB, pp. 372–383 (2004)
6. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. In: SODA, pp. 28–36 (2003)
7. Gil, Y., Kim, J., Puga, G.F., Ratnakar, V., González-Calero, P.A.: Workflow matching using semantic metadata. In: K-CAP, pp. 121–128 (2009)
8. Goderis, A., Li, P., Goble, C.A.: Workflow discovery: the problem, a case study from e-science and a graph-based solution. In: ICWS, pp. 312–319 (2006)
9. Kaufer, F., Klusch, M.: WSMO-MX: A logic programming based hybrid service matchmaker. In: ECOWS, pp. 161–170 (2006)
10. Kendall, M., Gibbons, J.D.: Rank Correlation Methods. Edward Arnold, London (1990)
11. Klusch, M., Fries, B., Sycara, K.P.: Automated semantic web service discovery with OWLS-MX. In: AAMAS, pp. 915–922 (2006)
12. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: WWW, pp. 331–339 (2003)
13. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5(1), 32–38 (1957)
14. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
15. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Comput. 27(7), 950–959 (2009)
16. Shao, Q., Sun, P., Chen, Y.: WISE: A workflow information search engine. In: ICDE, pp. 1491–1494 (2009)
17. Skoutas, D., Sacharidis, D., Simitsis, A., Sellis, T.: Ranking and clustering web services using multicriteria dominance relationships. IEEE T. Services Computing 3(3), 163–177 (2010)
18. Skoutas, D., Simitsis, A., Sellis, T.: A ranking mechanism for semantic web service discovery. In: IEEE SCW, pp. 41–48 (2007)