Efficient Semantic Web Service Discovery in Centralized and P2P Environments

Dimitrios Skoutas[1,2], Dimitris Sacharidis[1], Verena Kantere[3], and Timos Sellis[2,1]

[1] National Technical University of Athens
Athens, Greece
{dskoutas,dsachar}@dblab.ece.ntua.gr
[2] Institute for the Management of Information Systems (R.C. "Athena")
Athens, Greece
timos@imis.athena-innovation.gr
[3] Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
verena.kantere@epfl.ch

Abstract. Efficient and scalable discovery mechanisms are critical for enabling service-oriented architectures on the Semantic Web. The majority of currently existing approaches focuses on centralized architectures, and deals with efficiency typically by pre-computing and storing the results of the semantic matcher for all possible query concepts. Such approaches, however, fail to scale with respect to the number of service advertisements and the size of the ontologies involved. On the other hand, this paper presents an efficient and scalable index-based method for Semantic Web service discovery that allows for fast selection of services at query time and is suitable for both centralized and P2P environments. We employ a novel encoding of the service descriptions, allowing the match between a request and an advertisement to be evaluated in constant time, and we index these representations to prune the search space, reducing the number of comparisons required. Given a desired ranking function, the search algorithm can retrieve the top-k matches progressively, i.e., better matches are computed and returned first, thereby further reducing the search engine's response time. We also show how this search can be performed efficiently in a suitable structured P2P overlay network. The benefits of the proposed method are demonstrated through experimental evaluation on both real and synthetic data.

1 Introduction

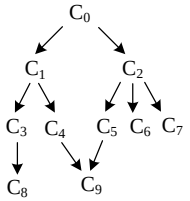
Web services enable interoperability and integration between heterogeneous systems and applications. Current industry standards for describing and locating Web services (WSDL, UDDI), describe the structure of the service interface and of the exchanged messages. Even though this provides interoperability at the syntactic level, it limits the discovery process to essentially keyword-based search. To increase the precision of the discovery process, appropriate services

should be identified and selected in terms of the semantics of the requested and offered capabilities. To that end, Semantic Web services combine the benefits of Semantic Web and Web services technology. Several approaches have been proposed for semantically enhancing the descriptions of Web services (OWL-S [1], WSDL-S [2], WSMO [3]), and automating the service discovery, composition, and execution. Service requests and advertisements are annotated by concepts from associated ontologies, and the matchmaking is based on subsumption reasoning between concepts corresponding to the requested and offered parameters.

As the number of services on the Web increases, the efficiency and the scalability of service discovery techniques become a critical issue. Moreover, several applications are inherently distributed. Consider, for example, a network of businesses or institutions, each providing its own services; creating and maintaining a centralized registry would not be desirable. However, the majority of current approaches focuses on centralized architectures, i.e., a single service registry or multiple service registries synchronizing periodically. This introduces bottlenecks and single points of failure, and fails to scale when the availability and demand for services grows significantly. On the other hand, P2P networks support large-scale, decentralized applications, offering scalability and reliability. In addition, structured P2P overlays provide guarantees for retrieving all search results in bounded time and distributing the load among peers. Hence, there has been recently a lot of research interest in issues overlapping the two fields, Semantic Web and P2P computing, primarily focusing on distributed RDF stores [4–6].

Regarding Semantic Web services, proposed approaches for service discovery in P2P environments typically rely on the use of ontologies to partition the network topology into concept clusters, and then forward requests to the appropriate cluster. However, constructing concept clusters in a fully automated way is not straightforward, as well as providing guarantees regarding search times and load balancing. In this paper we address the issue of Semantic Web service discovery, focusing on the aspects of efficiency and scalability. We present a method for fast search and selection of services at query time that is suitable for both centralized and P2P environments. In particular, our contributions are summarized in the following:

- We employ a novel encoding of the service descriptions, allowing the match between a service request and a service advertisement to be evaluated in constant time, avoiding the overhead of invoking the reasoner at query time.
- We index the service representations to prune the search space, minimzing the number of comparisons required to locate the matching services.
- We discuss the need for ranking the matched services, and present an algorithm that, given a desired ranking function, fetches the top-k matches progressively, thereby further reducing the search engine's response time.
- We extend our method to a structured P2P overlay network, showing that the search process can be done efficiently in a decentralized, dynamic environment.
- We demonstrate the efficiency and the scalability of our approach through experimental evaluation.



(a)

	INPUTS	OUTPUTS
R	C_8	C_4, C_7
$\mathbf{S}_1$	C_1	C_4, C_2
$\mathbf{S}_2$	C_3	C_9, C_7
$\mathbf{S}_3$	C_5	C_1

(b)

Concept	Intervals
C_0	[1,20]
C_1	[2,11]
C_2	[12,19],[8,9]
C_3	[3,6]
C_4	[7,10]
C_5	[13,14],[8,9]
C_6	[15,16]
C_7	[17,18]
C_8	[4,5]
C_9	[8,9]

(c)

Fig. 1: (a) A sample ontology fragment, (b) A service request (R) and three service advertisements (S_1, S_2, S_3), (c) Intervals assigned to ontology concepts

The rest of the paper is organized as follows. Section 2 discusses Semantic Web services matchmaking and ranking, and presents our encoding for service descriptions. Section 3 presents the indexing and searching of services in a centralized registry. Section 4 shows how service descriptions can be distributed and searched efficiently in a structured P2P overlay network. Experimental evaluation of the proposed approach is presented in Section 5, while related work is reviewed in Section 6. Finally, Section 7 concludes the paper.

2 The Matchmaking Framework

In this section we present our framework for efficient Semantic Web service matchmaking in centralized and P2P environments. First, we describe the service selection and ranking process in Section 2.1. Our framework is based on the encoding and indexing of service descriptions discussed in Section 2.2.

2.1 Semantic Selection of Services

In the following, we consider an ontology as a set of hierarchically organized concepts. Since multiple inheritance is allowed, the concepts form a rooted directed acyclic graph. The nodes of the graph correspond to concepts, with the root corresponding to the top concept, e.g., `owl:Thing` in an OWL ontology, whereas the edges represent subsumption relationships between the concepts, directed from the father to the child. To allow for semantic search of services on the Web, the description of a service is enhanced by annotating its parameters (typically inputs and outputs) with concepts from an associated ontology [1–3]. A service request is the description of a desired service, also annotated with ontology concepts. Figure 1a illustrates a sample ontology fragment, while a sample set of a service request and 3 service advertisements is shown in Figure 1b. The underlying assumption is that if a service provides as output (resp., accepts as input) a concept C, then it is also expected to likely provide (resp., accept) the

subconcepts of C. For instance, a service advertised as selling computers is expected to sell servers, desktops, laptops, PDAs, etc.; similarly, a service offering delivery in Europe is expected to provide delivery within all (or at least most) European countries.

Matchmaking of semantically annotated Web services is then based on subsumption reasoning between the semantic descriptions of the service request and the service advertisement. Along the lines of earlier works [7, 8], we specify the match between a service request R and a service advertisement S based on the semantic match between the corresponding parameters in their descriptions. More specifically, for a service parameter C_S and a request parameter C_R, we consider the match as *exact*, if C_S is equivalent to C_R ($C_S \equiv C_R$); *plug-in*, if C_S subsumes C_R ($C_S \sqsupset C_R$); *subsumes*, if C_S is subsumed by C_R ($C_S \sqsubset C_R$); *fail*, otherwise. Exact matches are preferable to plug-in matches, which in turn are preferable to subsumes matches. In the example of Figure 1, service S_1 provides one exact and two plug-in, service S_2 provides one plug-in, one subsumes, and one exact, whereas S_3 provides two fail and one plug-in matches.

Given that a large number of services may provide a partial match to the request, differentiating between the results within the same type of match is also required. Further following the aforementioned assumption regarding the semantics of a service description, we use as a criterion for assessing the degree of match between two concepts C_1 and C_2 the portion of their common subconcepts, or in other words, the extend to which the subtrees (more generally, subgraphs) rooted at C_1 and C_2 overlap. Intuitively, the higher the overlap, the more likely it is for the service to match the request. Thus, in the following, we consider the degree of match between two concepts C_1 and C_2 as

$$degreeOfMatch(C_1, C_2) = \frac{|\{C \mid C \sqsubseteq C_1 \wedge C \sqsubseteq C_2\}|}{\mathtt{max}(|\{C \mid C \sqsubseteq C_1\}|, |\{C \mid C \sqsubseteq C_2\}|)} \quad (1)$$

Returning to our example from Figure 1, notice that regarding the requested input, services S_1 and S_2 provide a plug-in match. However, using Equation (1), the degree of match for the service S_1 is 1/5, whereas for the service S_2 is 1/2. Notice that the proposed approach for service selection is not limited by this criterion. Different ranking criteria may be appropriate in different applications (for example, see [9] for a more elaborate similarity measure for ranking Semantic Web services). Our approach is generic and it can accommodate different ranking functions (see Section 3 for details). Retrieving services in descending order of their degree of match to the given request constitutes an important feature for a service discovery engine. In the case that the requester is a human user, it can be typically expected that he/she will navigate only the first few results. In fact, experiments conducted in a recent survey [10] showed that the users viewed the top-1 search result in about 80% of the queries, whereas results ranked below 3 were viewed in less than 50% of the queries. Even though this study refers to Web search, it is reasonable to assume a roughly similar behavior for users searching for services. On the other hand, Semantic Web service discovery plays an important role in fully automated scenarios, where a software agent, such as

a travel planning agent, acting on behalf of a human user, selects and composes services to achieve a specific task. Typically, the agent will select the top-1 match, ignoring the rest of the results. Hence, computing only the best possible match would be sufficient in this case. In fact, this often makes sense for human users as well; Google's "I'm Feeling Lucky" feature is a characteristic example based on this assumption.

2.2 Encoding of Service Descriptions

Invoking the reasoner to check for subsumption relationships between the ontology concepts annotating the service parameters constitutes a significant overhead, which has to be circumvented in order to allow for fast service selection at query time. For this purpose, we employ an appropriate service encoding based on labeling schemes [11]. The main idea works as follows. In the case of a tree hierarchy, each concept is labeled with an interval of the form $[begin, end]$. This is achieved by performing a depth-first traversal of the tree, and maintaining a counter, which is initially set to 1 and is incremented by 1 at each step. Each concept is visited twice, once before visiting any of its subconcepts and once after all its subconcepts have been visited. The interval assigned to the concept is constructed by setting its lower (resp., upper) bound to the value of the counter when the concept is visited for the first (resp., second) time. Observe that due to the way intervals are assigned, a concept C_1 is subsumed by another concept C_2 if and only if its interval is contained in that of C_2, i.e., $I_{C_1} \subset I_{C_2}$. This scheme generalizes to the case of graphs, which is the typical case for ontologies on the Semantic Web, by first computing a spanning tree T and applying the aforementioned process. Then, for each non spanning tree edge, the interval of a node is propagated recursively upwards to its parents. Therefore, more than one intervals may be assigned to each concept. As before, subsumption relationships are checked through interval containment: C_1 is subsumed by C_2 if and only if every interval of C_1 is contained in some interval of C_2.

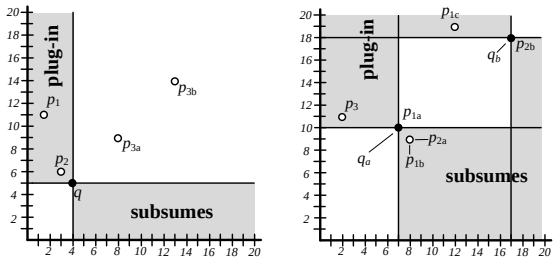
In our example, the intervals assigned to the ontology concepts are shown in Figure 1c, and have been computed considering the spanning tree formed by removing the edge (C_5, C_9). Notice how the interval assigned to the concept C_9 is then propagated to the concepts C_5, C_2, and C_0 (in the latter, it is subsumed by the initially assigned interval).

Consequently, a service request or advertisement can be represented by the set of intervals associated to its input and output concepts. With this encoding, determining the type of match between two service parameters is reduced to checking for containment relationship between the corresponding intervals; a constant time operation. In particular, we can rewrite the conditions determining the type of match between a request parameter C_R and a service parameter C_S, as shown in Table 1, where $\mathcal{I}_C$ denotes the set of intervals assigned to C.

Furthermore, the ranking criterion discussed in Section 2.2 can be expressed by means of the intervals based representation. For a concept C, the size of the subgraph rooted at C, G_C, is given by

Type of match	Condition
exact	$\mathcal{I}_{C_R} = \mathcal{I}_{C_S}$
plug-in	$\mathcal{I}_{C_R} \subset \mathcal{I}_{C_S}$
subsumes	$\mathcal{I}_{C_R} \supset \mathcal{I}_{C_S}$

Table 1: Types of match using the intervals based encoding



(a) Input parameters (b) Output parameters

Fig. 2: Interval based search

$$|G_C| = \sum_{I \in \mathcal{I}_C} \left\lceil \frac{|I|}{2} \right\rceil \tag{2}$$

Hence, for two concepts C_1, C_2, where $C_1 \sqsubseteq C_2$ or $C_1 \sqsupseteq C_2$, Equation 1 becomes:

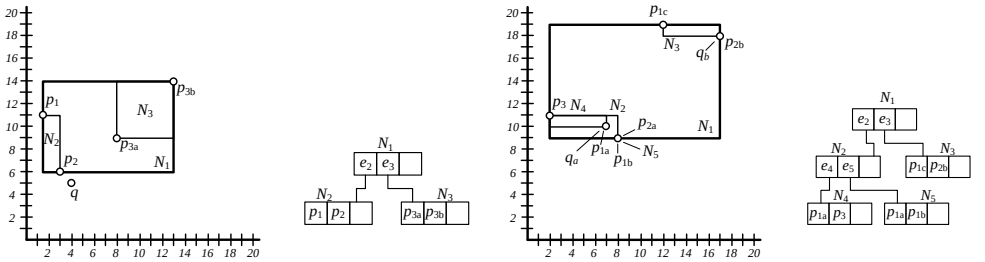
$$degreeOfMatch(C_1, C_2) = \frac{\min\{|G_{C1}|, |G_{C2}|\}}{\max\{|G_{C1}|, |G_{C2}|\}} \tag{3}$$

The above presented service representation allows the evaluation of the type and degree of match between a pair of requested and offered services in constant time. Still, the number of comparisons required is proportional to the number of available services. To further reduce the time required by the matcher, an index structure is employed for pruning the search space, keeping the number of comparisons required to a minimum. For this purpose, each interval is represented as a point in a 2-dimensional space, with the coordinates corresponding to the intervals' lower and upper bounds respectively, i.e., *begin* and *end*. Then, checking for containment between intervals is translated to a range query on this space. Figures 2a and 2b draw the input and output parameters, respectively, of the example in Figure 1. Points labeled as q_x, correspond to the parameters of the requested service, whereas p_{ix} correspond to parameters of the i-th offered service. For example, the output parameters of service S_2 is represented by points $p_{2a} = (8, 9)$ for class C_9 and $p_{2b} = (17, 18)$ for class C_7. For a given interval, the intervals contained by it are those located in its lower-right region, whereas those containing it are located in its upper-left region.

3 Centralized Service Discovery

In a centralized environment a single registry contains the information about all the advertised services and is responsible for performing the matchmaking and ranking process. Under our framework, this registry encodes all service descriptions and uses multi-dimensional indexes, such as the R-tree [12], to expedite service selection. The R-tree partitions points in hierarchically nested, possibly overlapping, *minimum bounding rectangles* (MBR). Each node in the tree stores a variable number of entries, up to some predefined maximum. Leaf nodes contain data points, whereas internal nodes contain the MBRs of their children.

We use two R-trees, T_{in}, T_{out} to index the services, where T_{in} (T_{out}) stores the intervals associated with the input (output) parameters. Consider as an example the 3 services discussed in Section 2.2. Figure 3 shows the MBRs and the structure of the two R-trees. An MBR is denoted by N_i and its corresponding entry as e_i. Notice that points that are close in the space (e.g., p_1, p_2 in Figure 3a) are grouped and stored in the same leaf node (N_2 in Figure 3b).



(a) T_{in} MBRs (b) T_{in} structure (c) T_{out} MBRs (d) T_{out} structure

Fig. 3: R-trees example

In the following we describe the algorithm (shown in Figure 4) for finding the services matching a request using our running example. The algorithm examines all request parameters in turn (Line 2). Assume that the first examined parameter *par* is the input corresponding to concept C_8; thus, T_{in} is examined (Line 3). The intervals, in this case $[4, 5]$, associated with the ontology concept is inserted in $\mathcal{I}$ (Line 5). Subsequently, three queries are posed to T_{in} retrieving the exact matches under point $(4, 5)$ (Line 7), the plug-in matches inside the range extending from $(0, 5)$ up to $(4, \infty)$ (Line 8) and the subsumes matches inside $(4, 0)$ up to $(\infty, 5)$ (Line 9). A range query is processed traversing the R-tree starting from the root. At each node, only its children whose MBR overlaps with the requested range are visited. Similarly, for the case of a point query, only children whose MBR contains the requested point are visited. A small performance optimization is to perform the three queries in parallel minimizing, thus, node accesses. Subsequently, all matches to *par* are merged into m_{par} (Line 10).

Once all parameters have been examined, the candidate services $\mathcal{S}_\mathcal{R}$ are constructed by intersecting the parameter matching results (Line 11). This retains only the services which match all request parameters. Since some services in $\mathcal{S}_\mathcal{R}$ can have additional input parameters that are not satisfied by the request, they are filtered out from the final result (Line 12).

Search Algorithm

Input: request R, available services $\mathcal{S}$ indexed in T_{in}, T_{out}
Output: services $\mathcal{S}_\mathcal{R}$ matching R

```
1  begin
2      foreach par ∈ IN_R ∪ OUT_R do
3          if par ∈ IN_R then T ← T_in
4          else T ← T_out
5          I ← the intervals associated with par
6          foreach interval I = (i_s, i_e) ∈ I do
7              m^ex_par ← point [i_s, i_e] query in T
8              m^pl_par ← range (0, i_e) × (i_s, ∞) query in T
9              m^sb_par ← range (i_s, 0) × (∞, i_e) query in T
10         m_par = m^ex_par ∪ m^pl_par ∪ m^sb_par
       S_R = ⋂_par m_par
11
12     S_R = S_R \ {S : ∃IN_S not matched by any IN_R}
13     return S_R
14 end
```

Fig. 4: Algorithm for index-based service matchmaking

As discussed in Section 2.1, in many cases a ranked list of the top-k best matching services is preferred as the result of the matchmaking process. Figure 5 illustrates the Progressive Search Algorithm to retrieve the top-k services given a request R using our framework. As before we present the algorithm using our running example. Initially, all intervals associated with the request parameters are inserted into $\mathcal{I}$ (Line 2). In particular, $\mathcal{I}$ contains $[4, 5]$ (represented by point q in Figure 3a) for the input parameter, and $[7, 10]$, $[17, 18]$ (represented by points q_a, q_b, respectively, in Figure 3c) for the two output parameters. A heap H_I is associated with each interval $I = [i_s, i_e] \in \mathcal{I}$ (Lines 3–7); in our case there are 3 heaps for q, q_a and q_b. Initially, these heaps contain the root node of T_{in} or T_{out}, depending on the interval's parameter type (Lines 5–6). Entries e_I in I's heap are R-tree nodes and are sorted increasingly by their *minimum distance* (MINDIST) to (i_s, i_e). The MINDIST of a leaf node, i.e., a point, is its distance from (i_s, i_e). The MINDIST of an internal node, i.e., an MBR, is the minimum distance of the MBR from (i_s, i_e).

```
Progressive Search Algorithm
   Input: request R, available services S indexed in T_in, T_out, k
   Output: services S_R matching R, in descending order of degree of match
 1 begin
 2     I ← the intervals associated with all parameters in IN_R ∪ OUT_R
 3     foreach I = [i_s, i_e] ∈ I do
 4         create a heap H_I
 5         if I corresponds to some par ∈ IN_R then insert in H_I root of T_in
 6         else insert in H_I root of T_out
 7         H_I entries are sorted increasingly by their MINDIST to (i_s, i_e)
 8     while k > 0 do
 9         find the heap H_I whose head entry has the minimum MINDIST
10         e_I ← pop(H_I)
11         if e_I is an internal node then insert in H_I all children of e_I
12         else
13             let S be the service corresponding to e_I
14             let par_I be the parameter corresponding to interval I
15             mark that S has a match for par_I
16             if S has matches for all parameters in IN_R ∪ OUT_R then
17                 insert S in S_R ;                          // S is a result
18                 k ← k − 1
19                 if k = 0 then return S_R
20 end
```

Fig. 5: Algorithm for progressively returning matches

The Progressive Search Algorithm proceeds examining heap entries until k services have been retrieved (Lines 8–19). The heap whose head entry has the minimum MINDIST is selected (Line 9). In our example both heaps for q_a and q_b have MINDIST 0 as their head entry (T_{out}'s root) contains both q_a and q_b; assume q_a's heap is selected. The entry (node N_1 in *out*) is popped from the heap (Line 10) and since it is an internal node all its children are inserted in the heap (Line 11). Then, the heaps are examined again and q_a's heap is selected, as node N_2 is in its head and has MINDIST 0. N_2 is popped and its children are inserted. Repeating the process once more, a leaf entry p_{1a} is popped, which corresponds to the first output parameter of service S_1 (Lines 13–14). We mark that S_1 has a match for a request parameter (Line 15). Then, S is checked if it has matches for all parameters, i.e., it is a result (Lines 16–19). The algorithm returns when k results have been found.

The output of the algorithm is the ranked service list S_2, S_1, S_3. Notice that S_2 has a subsumes match but it is ranked higher than S_1, having only exact and plug-in matches. Further, S_3 is included even though is has two fail matches. This is due to the fact that the MINDIST function described does not discriminate among points in different regions with respect to the point corresponding to a

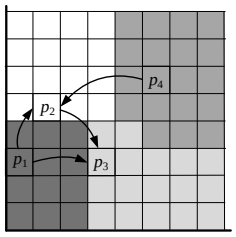


Fig. 6: Illustrative SpatialP2P overlay

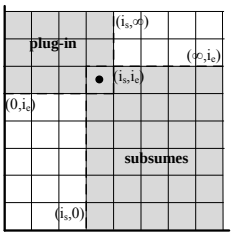


Fig. 7: Search regions

request parameter's interval. For example, in Figure 3c p_{2a} and p_{1b} are closer to q_a than p_3 and are regarded as better matches to parameter q_a, even though they are only subsumes matches (they lie in the lower right quadrant w.r.t. q_a). To obtain arbitrary rankings as described in Section 2.1, MINDIST can be trivially modified to be region aware. For example, it can evaluate heap entries that correspond to plug-in as closer compared to subsumes matches.

4 P2P Service Discovery

As the availability and demand for Web services grows, the issue of managing Semantic Web services in a distributed environment becomes vital. We describe a scalable and fault-tolerant solution that is adaptable and efficient in a distributed environment. From the variety of paradigms of distributed systems, we choose to focus on flat P2P overlays, as the latter represent the current trend for distributed data management. More specifically, we employ a structured P2P overlay, since it provides self-maintenance and robustness, as well as efficiency in data management. In the following we discuss the adaptation of our framework to the distributed setting.

4.1 The Underlying P2P Overlay

Before discussing distributed service discovery, we have to choose a suitable framework. To support the adaptation of the algorithms presented in Section 3, such a framework must support both point and range queries, so as to allow for the retrieval of both exact and plug-in/subsumes matches, respectively. Furthermore, since our proposed service encoding and service search algorithm are based on the 2-dimensional space, it is necessary to select a P2P framework that is efficient and scalable for 2-dimensional data. More specifically, the selected P2P framework should preserve locality and directionality, if possible.

SpatialP2P [13] is a recently proposed structured P2P framework, targeted to spatial data. It handles areas, which are either cells of a grid-partitioned space or sets of cells that form a rectangular. The basic assumption of the framework is that each area has knowledge of its own coordinates and the coordinates of some other areas to which it is directly linked. The goal of SpatialP2P is to

guarantee that any stored area can be searched and reached from any other, solely by exploiting local area knowledge.

Figure 6 shows an example of a SpatialP2P overlay with four peers. Each peer maintains links to others towards the four directions of the 2D space. The grid is hashed to the four peers, such that each cell is stored and managed by the closest peer. In the figure cells and their storing peers share the same color.

In SpatialP2P, search is routed according to locality and directionality. This means that search is propagated to the area that is closer to and towards the same direction with the sought area, choosing from the available areas that are linked to the one on which the search is currently iterated.

4.2 Managing Services in the P2P Overlay

The management of services in the P2P overlay consists of two basic operations: insertion of services and search for services in the system. Search can be either exhaustive, i.e., seeking for any possible results, or top-k, i.e., seeking the k best-matching results. In the following we discuss the details of these operations.

Service insertion. In order to use the P2P framework for the distributed management of Semantic Web services, we assume that the ID space of the overlay (i.e. the space of values for node and data IDs) corresponds to the space of values defined by the encoding of the service descriptions.

When a new service is published, its description is encoded using the intervals based representation presented in Section 2, and then it is inserted in the network. Specifically, each encoded service parameter is hashed to and eventually stored by the peer whose ID is closer to its value in the 2-dimensional space. Each inserted service parameter is accompanied by some meta-data about the respective type, (input or output), as well as the service it belongs to.

The locality-preserving property of the SpatialP2P overlay guarantees that similar services are stored by the same or neighboring peers. By similar, we mean services whose input and output parameters correspond to matching concepts. Moreover, the preservation of directionality means that following subsequent peers in a particular direction results, for example, in locating concepts subsuming or subsumed by the ones previously found. As described below, these properties are essential for minimizing the search time, and this applies to both exhaustive range and top-k queries.

Service search. Searching for services in the P2P overlay is performed by an adaptation of the search algorithm of Section 3 to the SpatialP2P API. For each requested service parameter, a point or a range query is performed, depending on the requirement of exact, plug-in or subsumes match with the available service parameters. An exact request for a service parameter corresponding to interval $I = [i_s, i_e]$ is performed by a point query asking the retrieval of the point (i_s, i_e), if such data exists in the overlay. For plug-in and subsumes requests for a parameter associated to the interval $I = [i_s, i_e]$, a pair of range queries is initiated. Since the data space is bounded (recall the intervals construction from Section 2.2), these requests are represented by range queries for

rectangular areas. Specifically, for plug-in matches, a query requesting the range extending from $(0, i_e)$ up to (i_s, ∞) is issued, while for the subsumes request, the corresponding range is $(i_s, 0) \times (\infty, i_e)$ (see Figure 7). The results of these two queries are unified to provide the answer to the requested parameter. Parallel searches are conducted for each requested parameter, and the results are finally intersected to compute the final matches.

Finding the top-k matches. SpatialP2P supports top-k search by extending search for range queries to dynamically increase the respective range. In detail, a search for a service parameter represented by an interval $I = [i_s, i_e]$ is initiated as the minimum range query that includes (i_s, i_e); thus, the minimum range is extended only in the grid cell in which the point (i_s, i_e) resides. After the search is performed in this minimum range, if the number of retrieved results is lower than k, then the range is increased towards the desired direction of the 2D space by the minimum, i.e., by one grid cell. The process repeats iteratively, until k results have been retrieved (or the whole space has been searched).

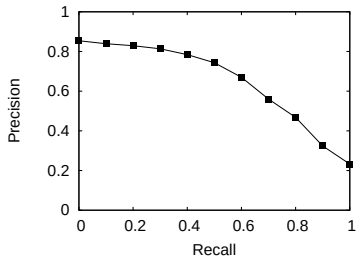
5 Experimental Evaluation

Experimental setup. We have evaluated our approach on two data sets. For the first data set, to simulate a real-world scenario, we used the OWL-S service retrieval test collection OWLS-TC v2[4]. This collection contains services retrieved mainly from public IBM UDDI registries, and semi-automatically transformed from WSDL to OWL-S. More specifically, it comprises: (a) a set of ontologies, derived from 7 different domains (education, medical care, food, travel, communication, economy and weapons), comprising a total of 3500 concepts, used to semantically annotate the service parameters, (b) a set of 576 OWL-S services, (c) a set of 28 sample requests, and (d) the relevance set for each request (manually identified).

The second data set was synthetically generated, based on the first one, so as to maintain the properties of real-world service descriptions. In particular, we constructed a set of approximately 10K services, by creating variations of the 576 services of the original data set. For each original service, we selected randomly one or more input or output parameters, and created a new service description by replacing them with randomly chosen superconcepts or subconcepts from the corresponding domain ontology. A set of 100 requests was generated following the same process, based on the original 28 requests. All the experiments were conducted on a Pentium D 2.4GHz with 2GB of RAM, running Linux.

Ranking. In the first set of experiments we used the first data set to evaluate the effectiveness of the ranking approach. For each of the 28 queries we retrieved the ranked list of match results, and compared them against the provided relevance sets. We use well-established IR metrics[5] to evaluate the performance of the search and ranking process. In particular, Figure 8a depicts the micro-averaged

[4] http://projects.semwebcentral.org/projects/owls-tc/
[5] http://trec.nist.gov/

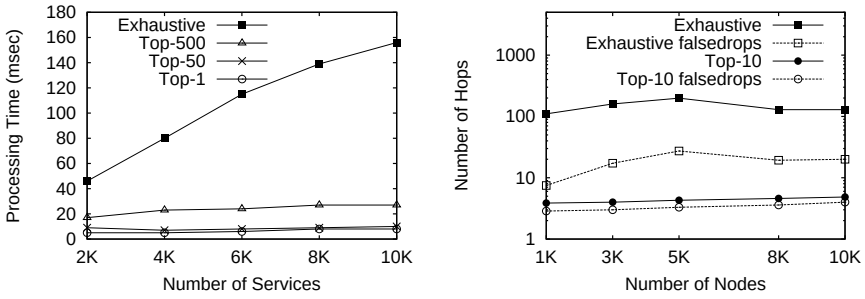


(a)

P@5	0.764
P@10	0.721
P@15	0.631
S@1	0.857
S@2	0.964
S@4	1

(b)

Fig. 8: (a) Recall-precision curve, (b) Precision and Success at k (P@k, S@k)



(a) Centralized (b) P2P

Fig. 9: Search cost for: (a) centralized registry, (b) P2P registry

recall-precision curves for all the 28 queries, i.e., the precision (averaged over all queries) for different recall levels. Observe that a 30% of the relevant services can be retrieved with precision higher than 80%, whereas for retrieving more than 70% of the relevant services the precision drops below 50%. Also, the following metrics are presented in Figure 8b: (a) precision at k, i.e., the (average) precision after k results have been retrieved; (b) success at k, i.e., whether a relevant result has been found after k results have been retrieved.

As we can see, the precision drops below 70% after the top-10 matches have been retrieved. Moreover, for the set of 28 queries, in 24 of them the top-1 match is a relevant one, in 27 queries there is a relevant result among the top-2 matches, and in all cases there is a relevant result among the top-4 matches. The above results opt for the emphasis on top-k queries and on fetching results progressively, as discussed in Section 2.

Experiments for centralized search. In this set of experiments we measured the time required by our search algorithm to discover and rank services in a centralized registry. In particular, we investigated the performance benefits, i.e., the reduction in response time, resulting from restricting the search to retrieving

only the top-k matches. For this purpose, we used the synthetically generated data set described previously. We varied the number of services from 2K up to 10K, and we measured the processing time (averaged over 100 queries) for retrieving: (i) all matches, and (ii) the top-k matches for $k \in \{1, 50, 500\}$. The experimental results are illustrated in Figure 9a. Notice the significant savings in the processing time when restricting the search to top-k matches, as well as the fact that the processing time in the latter case is significantly less sensitive to the number of available services.

Experiments for search in P2P environment. In the last set of experiments, we evaluated our search method in a P2P environment, as described in Section 4. We varied the size of the P2P network, from 1K up to 10K peers, and we inserted a total of 10K services. We conducted two experiments. In the first experiment, we retrieved, for each request, all the identified matches, whereas in the second, we restricted the search range to obtain (approximately) the top-10 results for each request. For each of the experiments we report two measures: (i) total number of hops (i.e., number of peers processing the query), and (ii) number of falsedrops (i.e., number of peers on the search path not contributing to the result set). The results are shown in Figure 9b. Both measures are quite low and relatively stable w.r.t. the size of the network. As discussed in Section 4, this is due to the fact that SpatialP2P is particularly designed to preserve the locality and the directionality of the data space, thus queries are effectively routed towards peers containing relevant information. As in the centralized case, the search cost is significantly lower, when retrieving only the top-k matches.

6 Related work

Service discovery is an important issue for the Semantic Web, hence several works have dealt with this problem. Matchmaking for Semantic Web services based on inputs and outputs has been studied in [7, 8], and more recently in [14, 15]. These form the basis of our matching approach, however they do not deal with the aspects of efficiency, ranking, and discovery in P2P networks, which are the main issues of our work. Implemented systems for matchmaking of OWL-S and WSMO services are described in [16, 17]. In [9, 18] similarity measures for ranking Semantic Web services are presented. These measures can be used by our top-k search algorithm, and hence are complementary to our work. The efficiency of the discovery process is considered in [19]. However, it is based on pre-computing and storing, for each concept in the ontology, the list of services matching this concept (together with the type of match). This imposes excessive storage requirements, and fails to scale as the number of available services (i.e., the size of the stored lists) and the size of the ontologies (i.e., the number of lists to store) grow significantly. Efficient matchmaking, together with ranked retrieval, is presented in [20]. Similar to our work, it uses intervals and indexing, which are however constructed in a different way. Moreover, search in P2P networks is not considered.

A P2P approach for Web service discovery is presented in [21]. However, the services are not semantically described; instead, the search is based on (possibly partial) keywords. Semantic Web service discovery in P2P networks has been studied in [22, 23]. In contrast to our work, these approaches deal with unstructured networks. In [24] Web service descriptions are indexed by keywords taken from domain ontologies, and are then stored on a DHT network. In [25] the peers are organized in a hypercube and the ontology is used to partition the network into concept clusters, so that queries are forwarded to the appropriate cluster. However, the subset of concepts to be used as structuring concepts should be known in advance. The approach in [26] distributes semantic service advertisements among available registries, by categorizing concepts into different groups based on their semantic similarity, and assigning groups to peers. In [27] services are distributed to registries depending on their type, e.g., a registry related to the travel domain will only maintain Web services specific to this domain.

7 Conclusions

We have presented and evaluated an efficient and scalable approach to Semantic Web service discovery and ranking. Efficiency is achieved by employing a suitable encoding for the service descriptions, and by indexing these representations to effectively prune the search space, consequently reducing the search engine's response time. To allow for scalability, we describe how the service representations can be distributed in a suitable structured P2P overlay network, and we show how the search is performed in this setting.

In this work we have treated the matching of service requests and advertisements as a matching of their inputs and outputs. However, service descriptions may also contain preconditions and effects, as well as QoS parameters. The described ideas can be easily extended to consider these additional criteria. In the future we plan to incorporate such parameters in our search algorithm.

References

1. Burstein, M., et. al.: OWL-S: Semantic Markup for Web Services. In: W3C Member Submission. (2004)
2. Akkiraju, R., et. al.: Web Service Semantics - WSDL-S. In: W3C Member Submission. (2005)
3. H. Lausen, A. Polleres, and D. Roman (eds.): Web Service Modeling Ontology (WSMO). In: W3C Member Submission. (2005)
4. Liarou, E., Idreos, S., Koubarakis, M.: Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In: ISWC. (2006) 399–413
5. Sidirourgos, L., Kokkinidis, G., Dalamagas, T., Christophides, V., Sellis, T.K.: Indexing Views to Route Queries in a PDMS. Distributed and Parallel Databases **23**(1) (2008) 45–68
6. Staab, S., Stuckenschmidt, H., eds.: Semantic Web and Peer-to-Peer. Springer Verlag (2006)

7. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic Matching of Web Services Capabilities. In: ISWC. (2002) 333–347
8. Li, L., Horrocks, I.: A Software Framework for Matchmaking based on Semantic Web Technology. In: WWW. (2003) 331–339
9. Skoutas, D., Simitsis, A., Sellis, T.: A Ranking Mechanism for Semantic Web Service Discovery. In: IEEE SCW. (2007) 41–48
10. Joachims, T., Radlinski, F.: Search Engines that Learn from Implicit Feedback. IEEE Computer **40**(8) (2007) 34–40
11. Christophides, V., Karvounarakis, G., Plexousakis, D., Scholl, M., Tourtounis, S.: Optimizing Taxonomic Semantic Web Queries Using Labeling Schemes. J. Web Sem. **1**(2) (2004) 207–228
12. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: SIGMOD Conference. (1984) 47–57
13. Kantere, V., Skiadopoulos, S., Sellis, T.: Storing and Indexing Spatial Data in P2P Systems. IEEE Transactions on Knowledge and Data Engineering (to appear).
14. Bellur, U., Kulkarni, R.: Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching. In: ICWS. (2007) 86–93
15. Skoutas, D., Sacharidis, D., Simitsis, A., Sellis, T.: Serving the Sky: Discovering and Selecting Semantic Web Services through Dynamic Skyline Queries. In: ICSC. (2008)
16. Klusch, M., Fries, B., Sycara, K.P.: Automated Semantic Web Service Discovery with OWLS-MX. In: AAMAS. (2006) 915–922
17. Kaufer, F., Klusch, M.: WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. In: ECOWS. (2006) 161–170
18. Cardoso, J.: Discovering Semantic Web Services with and without a Common Ontology Commitment. In: IEEE SCW. (2006) 183–190
19. Srinivasan, N., Paolucci, M., Sycara, K.P.: An Efficient Algorithm for OWL-S Based Semantic Search in UDDI. In: SWSWPC. (2004) 96–110
20. Constantinescu, I., Binder, W., Faltings, B.: Flexible and Efficient Matchmaking and Ranking in Service Directories. In: ICWS. (2005) 5–12
21. Schmidt, C., Parashar, M.: A Peer-to-Peer Approach to Web Service Discovery. WWW **7**(2) (2004) 211–229
22. Paolucci, M., Sycara, K.P., Nishimura, T., Srinivasan, N.: Using DAML-S for P2P Discovery. In: ICWS. (2003) 203–207
23. Basters, U., Klusch, M.: RS2D: Fast Adaptive Search for Semantic Web Services in Unstructured P2P Networks. In: ISWC. (2006) 87–100
24. Li, Y., Su, S., Yang, F.: A Peer-to-Peer Approach to Semantic Web Services Discovery. In: ICCS (4). (2006) 73–80
25. Schlosser, M.T., Sintek, M., Decker, S., Nejdl, W.: A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. In: P2P Computing. (2002) 104–111
26. Vu, L.H., Hauswirth, M., Aberer, K.: Towards P2P-Based Semantic Web Service Discovery with QoS Support. In: BPM Workshops. (2005) 18–31
27. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Inf. Tech. and Manag. **6**(1) (2005) 17–39