Request Rewriting-Based Web Service Discovery

Boualem Benatallah[1], Mohand-Said Hacid[2], Christophe Rey[3], and Farouk Toumani[3]

[1] SCSE, UNSW, Sydney, Australia
boualem@cse.unsw.edu.au
[2] LIRIS, UCB Lyon I, France
mshacid@liris.univ-lyon1.fr
[3] LIMOS, UBP, France
{rey,ftoumani}@isima.fr

Abstract. One of the challenging problems that Web service technology faces is the ability to effectively discover services based on their capabilities. We present an approach to tackle this problem in the context of DAML-S ontologies of services. The proposed approach enables to select the combinations of Web services that *best match* a given request Q and effectively computes the extra information with respect to Q (e.g., the information required by a service request but not provided by any existing service). We study the reasoning problem associated with such a matching process and propose an algorithm derived from hypergraphs theory.

Keywords: Semantic Web Services, Web Services Discovery, Description Logics, Hypergraphs

1 Introduction

Semantic Web services are emerging as a promising technology for the effective automation of services discovery, combination, and management [1,2,3]. Semantic Web services aim at leveraging two major trends in Web technologies, namely *Web services* and *Semantic Web*:

– Web services build upon XML as vehicle for exchanging messages across applications. The basic technological infrastructure for Web services is structured around three major standards: SOAP, WSDL, and UDDI [4,5]. These standards provide the building blocks for service description, discovery, and communication. While Web services technologies have clearly influenced positively the potential of the Web infrastructure by providing programmatic access to information and services, they are hindered by lack of rich and machine-processable abstractions to describe service properties, capabilities, and behavior. As a result of these limitations, very little automation support can be provided to facilitate effective discovery, combination, and management of services. Automation support is considered as the cornerstone to provide effective and efficient access to services in large, heterogeneous, and dynamic environments [6,4,3].

– Semantic Web aims at improving the technology to organise, search, integrate, and evolve Web-accessible resources (e.g., Web documents, data) by using rich and machine-understandable abstractions for the representation of resources semantics. Efforts in this area include the development of ontology languages such as RDF, DAML, and DAML+OIL [7].

By leveraging efforts in both Web services and semantic Web, semantic Web services paradigm promises to take Web technologies a step further by providing foundations to enable automated discovery, access, combination, and management of Web services. Efforts in this area focus on providing rich and machine understandable representation of services properties, capabilities, and behavior as well as reasoning mechanisms to support automation activities [1,8,2,3,9,10]. Examples of such efforts include the DAML-S [9] initiative and the WSMF-Web Services Modeling Framework [2]. Work in this area is still in its infancy. Many of the objectives of the semantic Web services paradigm, such as dynamic discovery and composition remain largely to be reached. Our work focuses on the issue of dynamic discovery of Web services. The notion of *dynamic discovery* (or simply service discovery) refers to systems in which requesters search through registries to discover and then invoke services supporting the capabilities they require.

This paper focuses on the reasoning aspects to automate the discovery of Web services. Our aim is to ground the discovery of Web services on a semantic comparison between a requester query and available Web services. We tackle this problem in the context of DAML-S service ontologies. In DAML-S, service requests as well as service capabilities are characterized, among others, in terms of their inputs and outputs. We propose a novel matching algorithm for discovering services. The matching algorithm takes as input a service request (or query) Q and a DAML-S ontology $\mathcal{T}$, and computes the best combination of Web services that satisfies *as much as possible* the outputs of the query Q and that requires *as little as possible* of inputs that are not provided in Q.

The main contributions of our approach are:

– We propose to view service discovery as a rewriting process in which a requester query Q is rewritten into the closest description E expressed as a conjunction of (some) Web services of a given ontology $\mathcal{T}$. Our approach features a global reasoning mechanism that goes beyond a simple pair-wise comparison between a service request and service offers.
– We propose a matching process that goes beyond simple subsumption tests between a service request and service advertisements.
  As emphasized in [11], a Web service discovery algorithm should support flexible matching since it is unrealistic to expect service requests and service offers to be exactly equivalent. To cope with this requirement, we propose to use a *difference operation* on service descriptions. Such an operation enables to extract from a subset of Web service descriptions the part that is semantically common with a given service request and the part that is semantically different from the request. Knowing the former and the latter allows to select relevant Web services and then to choose the best ones.

We investigate the reasoning problem associated to such a discovery process and its relationship with the expressiveness of the service description language. Then we propose a service discovery algorithm derived from hypergraphs theory. We show that this problem is similar to the so-called *best covering problem* investigated in [12,13]. In this paper we extend our previous results to deal with DAML-S ontologies of services.

The remainder of this paper is organized as follows: Section 2 motivates the role of ontologies to associate formal semantics with service descriptions and outlines the overall structure of a DAML-S ontology. Section 3 formalizes the proposed service discovery approach in the context of DAML-S ontologies. An overview of a service discovery algorithm is given in Section 4 and together with some preliminary experimental results are presented in Section 5. We review related work and give concluding remarks in Section 6.

2 Describing Web Services with DAML-S

Ontologies *"are formal and consensual specifications of conceptualisations that provide a shared and common understanding of a domain, an understanding that can be communicated across people and application systems"* [3]. The semantic Web community propose ontologies as means to address semantic heterogeneity among Web-accessible information sources and services. Ontologies are used to provide meta-data for the effective manipulation of available information including discovering information sources and reasoning about their capabilities. RDF, DAML, and DAML+OIL are examples of ontology languages [7]. In the area of Web services, ontologies promise to take interoperability a step further by providing rich description and modeling of services properties, capabilities, and behavior. DAML-S[1] is a DAML-OIL ontology for describing Web services. In this section, we give a brief overview of DAML-S as our approach relies on DAML-S to describe service properties and capabilities.

2.1 DAML-S Ontologies: An Overview

DAML-S employs the ontology structuring mechanisms of DAML+OIL to describe the properties and capabilities of Web services in a computer-interpretable form, thereby facilitating the automation of Web service discovery, invocation, composition and execution. As a DAML+OIL ontology, DAML-S has a well-defined semantics. It supplies a core set of markup language constructs for describing Web services in terms of classes (concepts) and complex relationships between them.

A DAML-S ontology of services is structured in three main parts [9]:

– *ServiceProfile* describes the capabilities and parameters of the service. It is used for advertising and discovering services.

[1] http://www.daml.org/services/

– *ServiceModel* gives a detailed description of a service's operation. Service operation is described in terms of a process model, which details both the control structure and data flow structure of the service required to execute a service.
– *ServiceGrounding* specifies the details of how to access the service, via messages (e.g., communication protocol, message formats, addressing, etc).

The service profile provides information about a service that can be used by an agent to determine if the service meets its needs. It consists of three types of information: a (human readable) *description* of the service; the *functional behavior* of the service which is represented as a transformation from the inputs required by the service to the outputs produced; and several *functional attributes* which specify additional information about a service (e.g., the cost of the service).

In the DAML-S approach, a service profile is intended to be used by providers to advertise their services as well as by service requesters to specify their needs. An example[2] of a very simple *book selling* service is given below.

2.2 Example

BookSellingService is a simple service which given a book title returns the price of that book. The advertisement below shows that the service accepts as input a string and generates as output instances of the concept Price as defined in a given ontology dummyOnt.

```
<profile:Profile rdf:ID="BookSellingService">
 <profile:serviceName>BookSellingService</profile:serviceName>
 ...
 <input>
  <profile:ParameterDescription rdf:ID="BookTitle">
    <profile:parameterName>bookTitle</profile:parameterName>
    <profile:restrictedTo rdf:resource="http://../XMLSchema.xsd#string"/>
  </profile:ParameterDescription>
 </input>
 <output>
  <profile:ParameterDescription rdf:ID="Price_Output">
    <profile:parameterName>Price</profile:parameterName>
    <profile:restrictedTo rdf:resource="dummyOnt.daml#Price"/>
  </profile:ParameterDescription>
 </output>
</profile:Profile>
```

A request for a service is expressed in the same format. For example, the previous profile description can also be used to express a request that looks for a service that accepts as input a string and generates as outputs instances of Price.

[2] This example is inspired from [11].

3 Characterizing Services Discovery Automation

In this section, we describe an approach for matching a service request with service advertisements. Given a service request (or query) Q and a DAML-S ontology $\mathcal{T}$ of services, we want to compute the best combinations of Web services that satisfy *as much as possible* the outputs of the request Q and that require *as little as possible* of inputs that are not provided in the description of Q. We call such combinations of Web services *best profile covers* of Q using $\mathcal{T}$. This problem is similar to the *best covering problem* [12,13,14]. We describe hereafter how the best covering problem can be extended to DAML-S ontologies.

3.1 Best Covering Profile Descriptions

To formally define the notion of *best profile cover* it is necessary to be able to characterize the notion of *"extra information"*, i.e., the information contained in one service profile description and not contained in the request description. For that, a *difference or subtraction* operation on service descriptions is required. As explained below, this requirement has an impact on the expressiveness of the ontology description language.

Characterizing the Ontology Description Language. Our approach uses Description Logics (DLs) [15] as a formal framework. DLs are a family of logics that were developed to model complex hierarchical structures and to provide a specialized reasoning engine to do inferences on these structures. Recently, DLs have heavily influenced the development of the semantic Web languages. For example, DAML+OIL, the ontology language used by DAML-S, is in fact an alternative syntax for a very expressive Description Logic [16].

A DL allows to represent domain of interest in terms of *concepts or descriptions* (unary predicates) that characterize subsets of the objects (*individuals*) in the domain, and *roles* (binary predicates) over such domain. Concepts are denoted by expressions formed by means of special constructors. Examples of DL constructors are given below:

- the symbol $\top$ is a concept description which denotes the top concept while the symbol $\bot$ stands for the inconsistent (bottom) concept,
- concept conjunction ($\sqcap$), e.g., the concept description $parent \sqcap male$ denotes the class of fathers (i.e., male parents),
- the universal role quantification ($\forall R.C$), e.g., the description $\forall child.male$ denotes the set of individuals whose children are all male.
- the number restriction constructor ($\geq n\ R$) e.g., the description ($\geq 1\ child$) denotes the class of parents (i.e., individuals having at least one children).

Let C, D be two concept descriptions. In the sequel, we use the expression $C \equiv D$ to denote equivalence between concept descriptions and $|C|$ to denote the size[3] of a concept C (i.e., the number of occurrences of concept and role names in C).

[3] Usually, $|C|$ is computed using a given canonical form of C.

We consider description logics with a difference operator between concept descriptions. Roughly speaking, the difference of two descriptions C and D, expressed using $C - D$, is defined as being a description containing all information which is a part of the description C but not a part of the description D [17]. However, it is worth noting that, in some description logics, $C - D$ may be a set of descriptions which are not semantically equivalent. Teege [17] provides sufficient conditions to characterize the logics where the difference operation is always semantically unique and can be implemented in a simple syntactical way by computing the set difference of subterms in a conjunction. According to [17], *structural subsumption* is a sufficient condition that allows to identify such logics. However, it is worth noting that the definition of structural subsumption given in [17] is different from the one usually used in the literature (e.g., see [15] for the usual definition). Nevertheless, the result given in [17] is still interesting in practice since there exist many description logics with this property. Examples of such logics include the quite expressive language $\mathcal{L}_1$ [17] , which contains the following constructors:

- $\sqcap, \sqcup, \top, \bot, (\geq\ n\ R)$, existential role quantification ($\exists R.C$) and existential feature quantification ($\exists f.C$) for concepts, where C denotes a concept, R a role and f a feature (i.e., a functional role),
- bottom ($\bot$), composition ($\circ$), differentiation ($|$) for roles,
- bottom ($\bot$) and composition ($\circ$) for features.

Currently, our approach does not consider the full expression power of the DAML+OIL language. We consider a subset of DAML+OIL for which a structural subsumption[4] algorithm exists. In the sequel, we use the term *restricted DAML-S ontologies* to denote DAML-S ontologies which are defined using sub-languages of DAML+OIL with structural subsumption.

The extension of our approach to consider more expressive description logics is currently under investigation. Such an extension requires the definition of a restricted difference operator to deal with the cases where the difference operation is not semantically unique.

Statement of the Problem. Now let us introduce some basic definitions that allow to extend the best covering problem to DAML-S service profiles.

Let $\mathcal{T} = \{S_i, i \in [1, n]\}$ be a restricted DAML-S ontology and $E \equiv S_l \sqcap \ldots \sqcap S_p$, with $l, p \in [1, n]$ and $l \leq p$, be a conjunction of some services occurring in $\mathcal{T}$. We denote by $I(E)$ (respectively, $O(E)$) the concept obtained by the conjunction of all the inputs (respectively, the outputs) occurring in the profile section of all the services S_i, for all $i \in [l, p]$. In the same way, we write $I(Q)$ (respectively, $O(Q)$) to denote the concept obtained by the conjunction of all the inputs (respectively, the outputs) occurring in the profile section of a given query Q.

Definition 1. ***Profile cover (Pcover)***
A profile cover, called Pcover, of Q using $\mathcal{T}$ is a conjunction E of some services S_i from $\mathcal{T}$ such that: $O(Q) - O(E) \not\equiv O(Q)$.

[4] In this paper we use the term *structural subsumption* in the sense of [17].

Hence, a Pcover of a query Q using $\mathcal{T}$ is defined as being any conjunction of Web services occurring in $\mathcal{T}$ that share some outputs with Q.

The following two definitions allow to characterize more precisely the remaining descriptions both in the outputs of the query Q (hereafter called the *Prest*) and in the inputs of its cover E (hereafter called the *Pmiss*).

Definition 2. *Profile rest (Prest) and Profile miss (Pmiss)*
Let Q be a service request and E be a Pcover of Q using $\mathcal{T}$.

- *The profile rest of Q with respect to E, written $Prest_E(Q)$, is defined as follows: $Prest_E(Q) \equiv O(Q) - O(E)$.*
- *The profile missing information of Q with respect to E, written $Pmiss_E(Q)$, is defined as follows: $Pmiss_E(Q) \equiv I(E) - I(Q)$.*

Now we can define the notion of best cover with respect to service profiles as follows.

Definition 3. *best profile cover*
A conjunction E of some services of an ontology $\mathcal{T}$ is called a best profile cover of Q using $\mathcal{T}$ iff:

- *E is a Pcover of Q using $\mathcal{T}$, and*
- *there doesn't exist a Pcover E' of Q using $\mathcal{T}$ such that $(|Prest_{E'}(Q)|, |Pmiss_{E'}(Q)|) < (|Prest_E(Q)|, |Pmiss_E(Q)|)$, where $<$ stands for the lexicographic order.*

The *best profile covering problem* is then defined as the problem of computing all the best profile covers of Q using $\mathcal{T}$. From the results already shown for the best covering problem [13], it follows that the best profile covering problem is NP-Hard.

3.2 Illustrating Example

This example illustrates how the notion of best profile cover can be used to match a service request with service advertisements. Let us consider an ontology of Web services containing the following three services:

- *ToTravel* allowing to reserve a trip given an itinerary (i.e., the departure point and the arrival point) and the arrival time and date.
- *FromTravel* allowing to reserve a trip given an itinerary and the departure time and date.
- *Hotel* allowing to reserve a hotel given a destination place, a period of time expressed in terms of the check-in date and the check-out date.

Due to lack of space, we do not provide the complete profile descriptions of these services. Table 1 shows the inputs and the outputs concepts of the three Web services. We assume that, the service profiles refer to concepts that are defined in the restricted[5] DAML-OIL ontology of tourism given in Table 2. For

[5] In this example, we use a description language made of the following constructors: concept conjunction ($\sqcap$), universal role quantification ($\forall R.C$) and the at least number restriction constructor ($\geq nR$).

Table 1. Input and Output service parameters.

Service	Inputs	Outputs
ToTravel	Itinerary, Arrival	TripReservation
FromTravel	Itinerary, Departure	TripReservation
Hotel	Destination, StayDuration	HotelReservation

Table 2. Example of a tourism ontology.

Itinerary $\equiv$	($\geq$ 1 departurePlace) $\sqcap$ ($\forall$ departurePlace.Location) $\sqcap$ ($\geq$ 1 arrivalPlace) $\sqcap$ ($\forall$ arrivalPlace.Location)
Arrival $\equiv$	($\geq$ 1 arrivalDate) $\sqcap$ ($\forall$ arrivalDate.Date) $\sqcap$ ($\geq$ 1 arrivalTime) $\sqcap$ ($\forall$ arrivalTime.Time)
Departure $\equiv$	($\geq$ 1 departureDate) $\sqcap$ ($\forall$ departureDate.Date) $\sqcap$ ($\geq$ 1 departureTime) $\sqcap$ ($\forall$ departureTime.Time)
Destination $\equiv$	($\geq$ 1 destinationPlace) $\sqcap$ ($\forall$ destinationPlace.Location)
StayDuration $\equiv$	($\geq$ 1 checkIn) $\sqcap$ ($\forall$ checkIn.Date) $\sqcap$ ($\geq$ 1 checkOut) $\sqcap$ ($\forall$ checkOut.Date)
TripReservation $\equiv$	...
HotelReservation $\equiv$	...
CarRental $\equiv$	...

clarity reasons, we use the usual DL syntax instead of the DAML-OIL syntax to describe the ontology. In Table 2, the description of the concept Itinerary denotes the class of individuals whose departure places (respectively arrival places) are instances of the concept Location. Moreover, the individual belonging to this class must have at least one departure place (the constraint ($\geq$ 1 departurePlace)) and at least one arrival place (the constraint ($\geq$ 1 arrivalPlace)). The input of the service ToTravel is obtained by the conjunction of all its inputs as follows: `I(ToTravel)` $\equiv$ `Itinerary` $\sqcap$ `Arrival`. By replacing the concepts Itinerary and Arrival by their descriptions, we obtain the following equivalent description:

`I(ToTravel)` $\equiv$ ($\geq$ 1 departurePlace) $\sqcap$ ($\forall$ departurePlace.Location) $\sqcap$ ($\geq$ 1 arrivalPlace) $\sqcap$ ($\forall$ arrivalPlace.Location) $\sqcap$ ($\geq$ 1 arrivalDate) $\sqcap$ ($\forall$ arrivalDate.Date) $\sqcap$ ($\geq$ 1 arrivalTime) $\sqcap$ ($\forall$ arrivalTime.Time)

The inputs and the outputs of the other Web services can be computed in the same way.

Now, let us consider a service request Q that looks for a vacation package that combines a trip with a hotel and a car rental, given a departure place, an arrival place, a departure date, a (hotel) destination place and the check-in and check-out dates. The inputs and outputs of the query Q can be expressed by the following descriptions which, again, refer to some concepts of the tourism ontology given in Table 2:

$I(Q)$ $\equiv$ ($\geq$ 1 departurePlace) $\sqcap$ ($\forall$ departurePlace.Location) $\sqcap$ ($\geq$ 1 arrivalPlace) $\sqcap$ ($\forall$ arrivalPlace.Location) $\sqcap$ ($\geq$ 1 departureDate) $\sqcap$ ($\forall$ departureDate.Date) $\sqcap$ ($\geq$ 1 destinationPlace) $\sqcap$ ($\forall$ destinationPlace.Location) $\sqcap$ ($\geq$ 1 checkIn) $\sqcap$ ($\forall$ checkIn.Date) $\sqcap$ ($\geq$ 1 checkOut) $\sqcap$ ($\forall$ checkOut.Date)

$O(Q)$ $\equiv$ TripReservation $\sqcap$ HotelReservation $\sqcap$ CarRental

The matching between the service request Q and the three advertised services given above can be achieved by computing the best profile covers of Q using these services. The result will be the following:

Best profile cover	**Prest**	**Pmiss**
FromTravel, Hotel	CarRental	departureTime

In this example, there is only one best profile cover of Q corresponding to the description $\mathtt{E} \equiv \mathtt{Hotel} \sqcap \mathtt{FromTravel}$. The selected services generate the concepts `TripReservation` and `HotelReservation` which are part of the output required by the query Q. From the service descriptions we can see that no Web service supplies the concept CarRental. Hence, the best profile covers of Q will have exactly the following profile rest: $\mathtt{Prest}_{\mathtt{E}}(\mathtt{Q}) \equiv \mathtt{carRental}$. This rest corresponds to the output of the query that cannot be generated by any advertised service. Moreover, the **Pmiss** column shows the information (the role departureTime) required as input of the selected services but not provided in the query inputs. More precisely, the best profile covers of Q will have exactly the following profile missing information: $\mathtt{Pmiss}_{\mathtt{E}}(\mathtt{Q}) \equiv (\geq 1\mathtt{departureTime}) \sqcap (\forall \mathtt{departureTime.Time})$. It is worth noting that, although the solution $\mathtt{E}' \equiv \mathtt{Hotel} \sqcap \mathtt{ToTravel}$ generates the same outputs (i.e., the concepts `TripReservation` and `HotelReservation`), it will not be selected because its **Pmiss** is greater than the one of the first solution (it contains the roles arrivalTime and arrivalDate).

4 Computing Best Profile Covers Using Hypergraphs

Let us first recall some useful definitions regarding hypergraphs. For more details about hypergraphs theory, we refer the reader to [18,19].

Definition 4. ***hypergraph and transversals*** *[19]*
An hypergraph $\mathcal{H}$ is a pair (Σ, Γ) of a finite set $\Sigma = \{V_1, \ldots, V_n\}$ and a set Γ of subsets of Σ. The elements of Σ are called vertices, and the elements of Γ are called edges.
A set $T \subseteq \Sigma$ is a transversal of $\mathcal{H}$ if for each $\varepsilon \in \Gamma$, $T \cap \varepsilon \neq \emptyset$. A transversal T is minimal if no proper subset T' of T is a transversal. The set of the minimal transversals of an hypergraph $\mathcal{H}$ is noted $Tr(\mathcal{H})$.

In the rest of this section, we briefly sketch how we can compute best profile covers using hypergraphs theory.

4.1 Reduction of the Best Profile Covering Problem to Hypergraphs

In the context of languages with structural subsumption, [17] shows that:

- each concept description C can be expressed in a given normal form, called Reduced Clause Form (RCF), as a conjunction of atomic clauses (e.g., $c_1 \sqcap \ldots \sqcap c_m$), and
- the difference $C - D$ between two concepts descriptions C and D can be computed using the simple *set difference* operation between the sets of atomic clauses of C and D.

Based on this result, we can show that the best profile covering problem can be interpreted in the framework of hypergraphs as the problem of finding the minimal transversals with a minimal cost. Given a query Q and an ontology of Web services $\mathcal{T}$, the first step is to build an associated hypergraph $\mathcal{H}_{\mathcal{T}Q}$ as follows:

– each Web service S_i in $\mathcal{T}$ becomes a vertex V_{S_i} in the hypergraph $\mathcal{H}_{\mathcal{T}Q}$.
– each clause A in the normal form description of the output $O(Q)$ of the query Q becomes an edge in the hypergraph $\mathcal{H}_{\mathcal{T}Q}$. The edge is populated by those services that have in their outputs a clause A' that is equivalent to A.
– Let $X = \{V_{S_i}, \ldots, V_{S_j}\}$ be a set of vertices of the hypergraph $\mathcal{H}_{\mathcal{T}Q}$. We define the notion of a cost of a set of vertices as: $cost(X) = |Pmiss_{(S_i \sqcap \ldots \sqcap S_j)}(Q)|$.

Then, one can prove that computing the best profile covers of Q using $\mathcal{T}$ can be reduced to the computation of the minimal transversals with the minimal cost of the associated hypergraph $\mathcal{H}_{\mathcal{T}Q}$. More details about the reduction of the best covering problem to hypergraphs can be found in [13].

4.2 The Service Discovery Algorithm

Based on hypergraph theory, we propose an algorithm called *computeBProfile-Cov* that solves the best profile covering problem. *computeBProfileCov* is an adaptation of the *computeBCov* algorithm proposed in [13] to solve the general best covering problem.

A naive approach to compute the minimal transversals with the minimal cost of a hypergraph would be: *(i)* to compute all the minimal transversals using existing algorithms for computing the minimal transversals of an hypergraph (e.g., see [18,19]), and then *(ii)* to choose those transversals which have the minimal cost. A simple algorithm to compute the minimal transversals of an hypergraph $\mathcal{H}$ is proposed in [20]. The algorithm is incremental and works in n steps where n is the number of edges of the hypergraph $\mathcal{H}$. Starting from an empty set of transversals, the basic idea is to explore each edge of the hypergraph, one edge at each step, and generate a set of candidate transversals by computing all the possible unions between the candidates generated in the previous step and each vertex in the considered edge. At each step, the non-minimal candidate transversals (i.e., the candidates in which at least one other candidate is strictly included) are removed.

The algorithm *computeBProfileCov* makes an improvement over the naive approach in two ways:

1. it reduces the number of candidates in the intermediary steps of the algorithm by generating only the minimal transversals, and
2. it uses a combinatorial optimization technique (Branch-and-Bound) in order to prune, at the intermediary steps of the algorithm, those candidate transversals which will not generate transversals with a minimal cost.

Algorithm 1 ***computeBProfileCov*** (sketch)

```
Require: An ontology $\mathcal{T}$ and a query $\mathcal{Q}$.
Ensure: The set of the best profile covers of $Q$ using $\mathcal{T}$.
 1: Build the associated hypergraph $\mathcal{H}_{\mathcal{T}Q} = (\Sigma, \Gamma)$.
 2: $Tr(\mathcal{H}_{\mathcal{T}Q}) \leftarrow \emptyset$     – Initialization of the set of the minimal transversals.
 3:
        $CostEval \leftarrow \sum_{e \in \Gamma'} \min_{V_{S_i} \in e} (|Pmiss_{S_i}(Q)|)$.     – Initialization of CostEval
 4: for all edge $E \in \Gamma$ do
 5:     $Tr(\mathcal{H}_{\mathcal{T}Q}) \leftarrow$ the new generated set of the minimal transversals.
 6:     Remove from $Tr(\mathcal{H}_{\mathcal{T}Q})$ the transversals whose costs are greater than $CostEval$.
 7:     Compute a more precise evaluation of $CostEval$.
 8: end for
 9: for all $X = \{V_{S_i}, \ldots, V_{S_n}\} \in Tr(\mathcal{H}_{\mathcal{T}Q})$ such that $|Pmiss_{S_i \sqcap \ldots \sqcap S_n}(Q)| = CostEval$
    do
10:     return the concept $S_i \sqcap \ldots \sqcap S_n$ as a best profile cover of $Q$ using $\mathcal{T}$.
11: end for
```

The first improvement, that allows to generate only good candidates (only minimal transversals) at each iteration (line 5 of the algorithm), uses a necessary and sufficient condition provided by Theorem 1 below. This condition characterizes a pair (X_i, s_j) that will generate a non minimal transversal at iteration i, where X_i is a minimal transversal generated at iteration $i-1$ and s_j is a vertex of the i^{th} edge.

Theorem 1. *Let $Tr(\mathcal{H}) = \{X_i, i = 1..m\}$ the set of minimal transversals for the hypergraph $\mathcal{H}$, and $E = \{s_j, j = 1..n\}$ an extra edge of $\mathcal{H}$. Assume $\mathcal{H}' = \mathcal{H} \cup E$. Then, we have : $X_i \cup \{s_j\}$ is a non-minimal transversal of $\mathcal{H}'$ $\Leftrightarrow$ it exists a minimal transversal X_k of $\mathcal{H}$ such that $X_k \cap E = \{s_j\}$ and $X_k \setminus \{s_j\} \subset X_i$.*

Details and proof of Theorem 1 are given in [21].

The second improvement consists in a Branch-and-Bound like enumeration of transversals. First, a simple heuristic is used to efficiently compute a cost of a *good* transversal (i.e., a transversal expected to have a small cost). This can be carried out by adding, for each edge of the hypergraph, the cost of the vertex that has the minimal cost. The resulting cost is stored in the variable $CostEval$ (line 3 of the algorithm). As we have, for any set of vertices $X = \{V_{S_i}, \ldots, V_{S_n}\}$:

$$cost(X) = |Pmiss_{S_i \sqcap \ldots \sqcap S_n}(Q)| \leq \sum_{j \in [i,n]} |Pmiss_{S_j}(Q)| = \sum_{V_{S_j} \in X} cost(V_{S_j})$$

Hence, the evaluation $CostEval$ is an upper bound of the cost of an existing transversal. Then as we consider candidates in intermediate steps of the algorithm, we can eliminate from $Tr(\mathcal{H}_{\mathcal{T}Q})$ any candidate transversal that has a greater cost than $CostEval$, since that candidate cannot possibly lead to a transversal that is better than what we already know (line 6). Then, from each candidate transversal that remains in $Tr(\mathcal{H}_{\mathcal{T}Q})$, we compute a new evaluation for $CostEval$ by considering only remaining edges (line 7).

These two optimizations[6] have been implemented as separate options of the algorithm, namely option Pers for the first optimization and option BnB for the second one.

Hence, the algorithm *computeBProfileCov* can be used to allow dynamic discovery of DAML-S services based on their capabilities. According to the definitions 1 and 2 of Section 3, the algorithm selects the combinations of services that best match a given query and effectively computes the outputs of the query that cannot be satisfied by the available services (i.e., Prest) as well as the inputs that are required by the selected services and are not provided in the query (i.e., Pmiss).

5 Implementation and Experiments

To conduct experiments, we have developed a testbed prototype that allows to evaluate the performance of the *computeBProfileCov* algorithm. This prototype is implemented using the Java programing language. The prototype implements up to 6 versions of the *computeBProfileCov* algorithm corresponding to different combinations of optimization options. Moreover, it includes a tool that enables to generate random XML-based services ontologies and associated service requests that can be used to evaluate the *computeBProfileCov* algorithm. All experiments have been performed on a PC with a Pentium III 500 MHz and 384 Mo of RAM.

To test *computeBProfileCov*, we first have run *computeBProfileCov* on worst cases and then on a set of ontologies and queries randomly generated by our prototype. *computeBProfileCov* worst cases were built according to a theoretical study of the complexity of all versions of *computeBProfileCov*: two ontologies (and their associated query) have been built in order to maximize the number of minimal transversals of the corresponding hypergraph as well as the number of elementary operations of the algorithm (i.e., inclusion tests and intersection operations). In each case, there exists at least one version of *computeBProfileCov* that complete the execution in less than 20 seconds. We point out the fact that if these cases are very bad for *computeBProfileCov*, they are also totally unrealistic with respect to practical situations. Due to space limitation, we will not detail here the many parameters used in the generation of random ontologies. We just point out that we tried to generate larger but still realistic random ontologies. The modeling and random generating question is an hard issue that is far out from the scope of this paper. We focus here on three study cases which special feature is the varying size of the application domain ontology, of the Web service ontology and of the query. Their characteristics are given below:

[6] Other less significant optimization options have also been implemented.

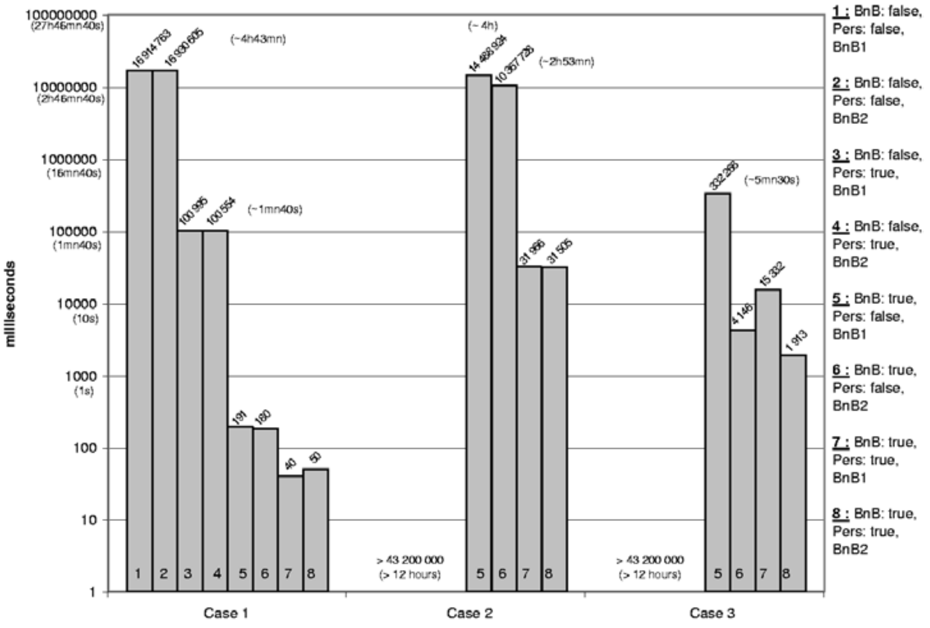

Fig. 1. Execution time

Configurations	**Case 1**	**Case 2**	**Case 3**
Number of concept descriptions in the application domain ontology	365	1334	3405
Number of Web services	366	660	570
Number of (atomic) clauses in the query	6	33	12

It is worth noting that the internal structures of these ontologies correspond to bad cases for the *computeBProfileCov* algorithm. We have run the 6 versions of the *computeBProfileCov* algorithm on each of these cases. The overall execution time results are given in Figure 1[7]. The Figure 1 shows that for the cases 1 and 3, there is at least a version of the algorithm that runs in less than two seconds, and in less than 30 seconds for the case 2. So, although the Figure 1 shows that there is a great difference in performance of the different versions of the algorithm, there is in each case one version of *computeBProfileCov* that allows us to be optimistic concerning the real-time execution of services discovery with quite large domain ontologies. All details concerning the implementation of *computeBProfileCov*, the theoretical study of worst cases, the parameterized ontology generation process and experimental results can be found in [21].

[7] Note that versions 1 and 2 of the algorithm (respectively, 3 and 4) are similar as both run *computeBProfileCov* without BnB, and what distinguishes 1 from 2 (respectively, 3 from 4) is the way the option BnB is implemented (BnB1 or BnB2).

6 Related Work and Concluding Remarks

There are some proposals of Web service discovery mechanisms that employ semantic Web technology for service description. For example, Bernstein et al. [10] proposes to use process ontologies to describe the behaviour of services and then to query such ontologies using a Process Query Language (PQL). Chakraborty et al. [8] defines an ontology based on DAML [22] to describe mobile devices and proposes a matching process, based on an advanced Prolog reasoning engine, to locate devices on the basis of their features. González-Castillo et al. [23] reports on an experience in building matchmaking prototype based on a description logic reasoning engine and operating on service descriptions in DAML+OIL. The proposed matching algorithm is based on simple subsumption and consistency tests. Paolucci et al. [11] proposes a matching algorithm between services and requests described in DAML-S. In this approach, a match between a service request and an advertised service is determined by comparing all the outputs of the query with the outputs of the advertisement and all the inputs of the advertisement with the inputs of the query. The proposed algorithm recognizes various degrees of matching that are determined by the minimal distance between concepts in the concept taxonomy. Based on a similar approach, the ATLAS matchmaker [24] operates on DAML-S ontologies and utilizes two separate sets of filters: *1)* Matching functional attributes to determine the applicability of advertisements (i.e., do they deliver sufficient quality of service, etc). *2)* Matching service functionalities to determine if the advertised service matches the requested service. A DAML-based subsumption inferential engine is used to compare input and output sets of requests and advertisements.

Finally, it should be noted that the problem of capabilities based matching has also been addressed by several other research communities, e.g., information retrieval, software reuse systems or multi-agent communities. More details about these approaches and their applicability in the context of the semantic Web service area can be found in [10,11].

Our work focuses on automatic discovery of Web services based on a semantic matching between declarative descriptions of services and requests. We adopt an approach similar to [11] for comparing requests with advertised services based on their inputs and outputs. However, we propose a rather different matching algorithm. Indeed, since we view the Web service discovery as a rewriting process, our algorithm is able to discover *combinations* of services that best match (cover) a given request. Furthermore, the difference between the query and its rewriting (i.e., Prest and Pmiss) is effectively computed and can be used to improve the Web service interoperability.

From the technical point of view, the best profile covering problem belongs to the general framework for *rewriting using terminologies* provided in [25]. This framework is defined as follows: given a terminology $\mathcal{T}$ (i.e., a set of concept descriptions), a concept description Q that does not contain concept names defined in $\mathcal{T}$ and a binary relation ρ between concept descriptions, can Q be rewritten into a description E, built using (some) of the names defined in $\mathcal{T}$, such that $Q\rho E$?

Additionally, some optimality criterion is defined in order to select the relevant rewritings. Already investigated instances of this problem are the minimal rewriting problem [25] and rewriting queries using views (e.g., [26,27]). In the former, ρ is instantiated by equivalence modulo $\mathcal{T}$ and the size of the rewriting is used as the optimality criterion. In the latter, the relation ρ is instanciated by subsumption and the optimality criterion is the inverse subsumption [25].

The *best profile covering problem* can be viewed as a new instance of the problem of rewriting concepts using terminologies where the goal is to rewrite a description Q into the closest description expressed as a conjunction of (some) concept names in $\mathcal{T}$ (hence, ρ is neither equivalence nor subsumption).

Our future work will be devoted to the extension of the proposed framework to hold the definition of the best profile covering problem for description logics where the difference operation is not semantically unique. After the very first results we got in this context, we argue that a restricted difference operator can be defined, and then the proposed framework can be extended to more expressive languages.

References

1. McIlraith, S., Son, T., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems. Special Issue on the Semantic Web **16** (2001) 46–53
2. Fensel, D., Bussler, C., Maedche, A.: Semantic Web Enabled Web Services. In: International Semantic Web Conference, Sardinia, Italy. (2002) 1–2
3. Fensel, D., Bussler, C., Ding, Y., Omelayenko, B.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications **1** (2002)
4. Casati, F., Shan, M.C., Georgakopoulos, D.G., (editors): The VLDB Journal: Special Issue on E-Services. 10(1), Springer-Verlag Berlin Heidelberg (2001)
5. Weikum, G., (editor): Data Engineering Bulletin: Special Issue on Infrastructure for Advanced E-Services. 24(1), IEEE Computer Society (2001)
6. Casati, F., Shan, M.C.: Dynamic and adaptive composition of e-services. Information Systems **26** (2001) 143–163
7. Ding, Y., Fensel, D., M.C.A. Klein, B.O.: The semantic web: yet another hip? DKE **6** (2002) 205–227
8. Chakraborty, D., Perich, F., Avancha, S., Joshi, A.: DReggie: Semantic Service Discovery for M-Commerce Applications. In: Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposium on Reliable Distributed Systems. (2001) 28–31
9. The DAML Services Coalition: DAML-S: Web Service Description for the Semantic Web. In: The 1st International Semantic Web Conference (ISWC). (2002) 348–363
10. Bernstein, A., Klein, M.: Discovering Services: Towards High Precision Service Retrieval. In: CaiSE workshop on Web Services, e-Business, and the Semantic Web: Foundations, Models, Architecture, Engineering and Applications. Toronto, Canada. (2002)
11. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matching of Web Services Capabilities. In: Semantic Web Conference, Sardinia, Italy. (2002) 333–347
12. Hacid, M., Leger, A., Rey, C., Toumani, F.: Computing concept covers: A preliminary report. In: International Workshop on Description Logics (DL 2002). Toulouse, France. (2002)

13. Hacid, M.S., Leger, A., Rey, C., Toumani, F.: Dynamic Discovery of E-services (A Description Logics Based Approach). Report, LIMOS, Clemont-Ferrand, France (2002) see http://www.710.univ-lyon1.fr/∼dbkrr/mshacid/publications.html.
14. Benatallah, B., Hacid, M.S., Rey, C., Toumani, F.: Semantic Reasoning for Web Services Discovery. In: WWW Workshop on E-Services and the Semantic Web, Budapest, Hungary. (2003)
15. Donini, F., M. Lenzerini, D. Nardi, A.S.: Reasoning in description logics. In: Gerhard Brewka, editor, Foundation of Knowledge Representation, CSLI-Publications (1996) 191–236
16. Horrocks, I., P.F.Patel-Schneider, van Harmelen, F.: Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web. In: Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002). (2002)
17. Teege, G.: Making the difference: A subtraction operation for description logics. In Doyle, J., Sandewall, E., Torasso, P., eds.: KR'94, San Francisco, CA, Morgan Kaufmann (1994)
18. Berge, C.: Hypergraphs. Volume 45 of North Holland Mathematical Library. Elsevier Science Publishers B.V. (North-Holland) (1989)
19. Eiter, T., Gottlob, G.: Identifying the minimal transversals of a hypergraph and related problems. SIAM Journal on Computing **24** (1995) 1278–1304
20. Mannila, H., Räihä, K.J.: The Design of Relational Databases. Addison-Wesley, Wokingham, England (1994)
21. Rey, C., Toumani, F., Hacid, M.S., Leger, A.: An algorithm and a prototype for the dynamic discovery of e-services. Technical Report RR05-03, LIMOS, Clemont-Ferrand, France, http://www.isima.fr/limos/publications.htm (2003)
22. Hendler, J., McGuinness, D.L.: The DARPA Agent Markup Language. IEEE Intelligent Systems **15** (2000) 67–73
23. González-Castillo, J., Trastour, D., Bartolini, C.: Description Logics for Matchmaking of Services. In: KI-2001 Workshop on Applications of Description Logics Vienna, Austria. (2001)
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-44/.
24. Payne, T., Paolucci, M., Sycara, K.: Advertising and Matching DAML-S Service Descriptions (position paper). In: International Semantic Web Working Symposium, Stanford University, California, USA. (2001)
25. Baader, F., Küsters, R., Molitor, R.: Rewriting Concepts Using Terminologies. In: Proc. of KR'00, Colorado, USA. (2000) 297–308
26. Beeri, C., Levy, A., Rousset, M.C.: Rewriting Queries Using Views in Description Logics. In Yuan, L., ed.: Proc. of the ACM PODS , New York, USA. (1997) 99–108
27. Goasdoué, F., V. Lattès, M.C.R.: The Use of CARIN Language and Algorithms for Information Integration: The PICSEL System. IJICIS **9** (2000) 383–401