Two-phased Web Service Discovery

Ruben Lara
Tecnologia, Informacion y Finanzas (TIF)
Grupo Analistas
rlara@afi.es

Abstract

Service-Oriented Architectures enable the consumption of existing services in a standardized way to solve some specific needs. In order to consume some services made available by cooperating parties e.g. for performing some business activity, such services must be located based on different criteria. In this paper, we investigate how the semantic description of the capability of Web services and of customer goals can be exploited for (semi)automating Web service discovery based on functional requirements. For this purpose, we propose a two-phased Web service discovery model. In a first phase, relevant Web services in terms of the results they potentially provide are efficiently located. In a second phase, the inputs required by relevant Web services are considered and only Web services for which the customer can provide appropriate input and for which this input can lead to the expected results are selected. The first phase relies on subsumption reasoning with Description Logics, while the second one relies on a combination of query answering for Logic Programs and subsumption reasoning with Description Logics.

Introduction

Service-Oriented Architectures (SOAs) have emerged as a paradigm for building scalable and easy-to-integrate IT systems, relying on the existence of services that perform some business function and that can be consumed in a standardized fashion. This paradigm is gaining momentum, and the common goal of many companies is to implement an enterprise-wide SOA that crosses departmental and geographical boundaries. Furthermore, special interest is paid to the extension of the service-oriented view to business collaboration, crossing enterprise boundaries as well. The implementation of an enterprise-wide SOA already leads to an scenario where an important number of business services are available for their stand-alone or composed exploitation. This requires advanced facilities for locating services that can provide some particular business function required for the task at hand. We believe that the semantic description of Web services[1] can help to (semi)automate the discovery of Web services that perform some relevant business function, avoiding the bottleneck that the increase in the number of available Web services can create in the location and use of services in an SOA.

In this paper, building on the work presented in (Keller *et al.* 2005), we propose a two-phased model for Web service discovery based on the formalization of the functional requirements of a customer and of the functional offer of available Web services. The first phase efficiently identifies Web services that can provide results required by the customer. In the second phase, the input values required by relevant Web services are considered and only Web services for which the customer can provide appropriate input and for which this input can lead to the expected results are selected.

Use case

The use case that will be used in the remainder of the paper is based on the following scenario: an end-user or a business process needs: a) to find investment funds meeting certain criteria e.g. commercialized by a given entity, or b) to purchase a given investment fund. Different Web services that provide information on investment funds exist, offering information about a different set of funds, a different set of information from each fund, and requiring different inputs. Investment funds can also be purchased using different Web services. The Web service discovery process will locate those Web services that can be used to request the information required or to purchase the desired fund. Throughout the paper we will use the following set of goals and Web services in order to illustrate our approach:

- Goal 1 (G1): descriptive information about all investment funds from Spanish institutions is required.
- Goal 2 (G2): descriptive information about all funds from Spanish institutions and categorized as FIAMM-CNMV (a category defined by the Spanish market supervisor) is required, as well as their ranking in this category.
- Goal 3 (G3): shares from the investment fund *myFund* have to be purchased and paid with credit card *myCC*.
- Web service 1 (WS1): can provide descriptive information about all funds from European institutions. It requires

[1] In the remainder of the paper we will focus on services exposed via Web service interfaces. However, the approach presented is equally applicable to other technologies such as JMS.

a European country as input, and gives information about all funds from that country.

- Web service 2 (WS2): can provide descriptive information about all funds from Spanish institutions. It requires a Spanish institution as input, and gives information about all funds from that institution.
- Web service 3 (WS3): can provide descriptive information about all funds from Spanish institutions, as well as their ranking. It requires a funds category as input, and provides the Spanish funds classified under that category as well as their ranking in such category.
- Web service 4 (WS4): can provide the purchase of shares of any European investment fund with a credit card. It requires the European fund to be purchased and the card details as input, as well as the details of a registered user of the service, and provides the purchase of the fund with the card given.

The paper is structured as follows: in the next section, we introduce the concepts of service, Web service and goal in order to have a common understanding of these terms. Afterwards, we present the type of formal descriptions of goals and Web service functional capabilities that can be generally expected. The Web service discovery process proposed is described next, including the restrictions introduced on the expressivity of the language used for describing goals and capabilities. Finally, we present our conclusions and conduct a brief discussion.

Services, Web services, and goals

In this section we present our understanding of the concepts of service, Web service, and goal, based on the work in (Keller *et al.* 2005).

Services

We understand a service (or business service) as a provision of value in some domain (Preist 2004). Let us consider G3: the service sought is the purchase of shares from a given investment fund with a given credit card, but the purchase itself is independent on how supplier and provider interact i.e. it does not matter whether the customer goes to a bank office to request the purchase of the fund or uses a Web service from an institution commercializing the fund. We understand the term service in this sense, that is, as a provision of value, irrespective of how customer and provider interact.

The provision of a service $serv$ causes some changes on the state of the world, called *service effects*, and the provision of some information to the customer, called *service outputs* (Keller *et al.* 2005). An example of the former is the purchase of some shares of an investment fund and the associated charge of the customer credit card, while an example of the latter is the provision of descriptive information about funds from Spanish institutions.

Web services

A Web service is a computational entity accessible using some particular standards and protocols, and usable to request some services. If we consider a Web service like WS4, we can see that it is an electronic means to request a service i.e. a purchase, but not the service itself.

An *abstract service* $\mathcal{A}^{\mathcal{W}}$ is the collection of services that can be requested using a Web service $\mathcal{W}$. In particular, an abstract service comprises a set of services $Serv_{\mathcal{A}^{\mathcal{W}}} = \{serv_1, \ldots, serv_n\}$ and, therefore, a set of outputs and effects resulting from the provision of each service $serv_i \in Serv_{\mathcal{A}^{\mathcal{W}}}$. For example, the abstract service provided by WS4 is the purchase of shares from European investment funds with credit card. The constituent services are each concrete purchase that can be requested via the Web service, for example the purchase of a concrete fund with a given credit card by a given investor.

A Web service will generally require some input information to be sent to it. The concrete service $serv_i \in Serv_{\mathcal{A}^{\mathcal{W}}}$ provided by $\mathcal{W}$ and, therefore, the outputs and effects resulting from invoking $\mathcal{W}$, will depend on the state of the world s that holds when $\mathcal{W}$ is used to request a service, and on the set of input values $i_1, \ldots, i_n$ given by the requester to $\mathcal{W}$. For example, the concrete service provided by WS4 i.e. the concrete purchase is dependent on the fund and credit card details given as input. The current state says e.g. whether the fund requested is being currently commercialized and, therefore, whether the purchase is possible at all.

In general, we can see a Web service $\mathcal{W}$ as a function that maps a particular set of input values given by the requester and a particular state of the world to a service $serv_i \in Serv_{\mathcal{A}^{\mathcal{W}}}$ and, in turn, to the outputs and effects of $serv_i$. However, not all states and input values will lead to the provision of some service by $\mathcal{W}$, but only states fulfilling some conditions defined by Web service *preconditions*, and input values satisfying the information requirements of the Web service as defined by Web service *inputs*, will lead to the provision of a service $serv_i \in Serv_{\mathcal{A}^{\mathcal{W}}}$ i.e. the domain of the function the Web service offers is restricted to states and input values satisfying the Web service preconditions and input requirements.

Goals

A customer will request the provision of a service (via a Web service) because it has some particular functional value. In particular, the goal of a customer when requesting the provision of a service is to know some new information and/or to make something hold in the real world. A goal expresses the results that are expected from the provision of a service.

If we take G1 as an example, the functional value expected is the provision of certain information about funds meeting certain criteria, while if we consider G3 the functional value expected is the actual purchase of shares of a given investment fund with credit card i.e. some effects on the real world are expected from the service provision.

Semantic descriptions

For (semi)automating the Web service discovery process, it is necessary to count with a formal description of customers' goals and of the functional value of available Web services. In this section, we will introduce the generic types of descriptions we expect to have available during the discovery

process. The descriptions introduced in this section will make use of full first-order logic (Fitting 1996), with classical first-order semantics. The purpose is to illustrate the type of descriptions that are considered. In the next section, however, restrictions on the expressivity allowed will be introduced and local closed world reasoning will be used.

Goal descriptions

The goal of a customer when requesting the provision of a service via a Web service is to gather some information and/or to make some conditions hold in the real world. Therefore, a goal $\mathcal{G}$ can be described by the set of results, both outputs and effects, that are expected from the provision of a service. These results can be formalized by a first-order predicate $res_{\mathcal{G}}(x)$ defined as:

$$\forall x.res_{\mathcal{G}}(x) \leftrightarrow \phi_{\mathcal{G}}(x)$$

where $\phi_{\mathcal{G}}(x)$ denotes a first-order formula over variable x that defines what conditions an object x must fulfill for being an expected result and, viceversa, what conditions expected results fulfill. Intuitively, $res_{\mathcal{G}}(o)$ will be true for an object o iff o is an expected result for achieving goal $\mathcal{G}$. For example, predicate $res_{G3}(x)$ describing the results expected by G3 would be defined as follows:

$$\forall x.res_{G3}(x) \leftrightarrow purchase(x) \wedge ofItem(x, myFund) \wedge withPaymentMethod(x, myCC)$$

where *myFund* and *myCC* are constants denoting a given investment fund and credit card, and $purchase$, $ofItem$ and $withPaymentMethod$ are predicates defined by an underlying domain model that provides the necessary vocabulary for the description of goals in a particular domain. Notice that, for simplicity, we consider a single predicate for any kind of results; extending the model for having two predicates, one for outputs and another one for effects, is straightforward.

Customer information. Web services will generally require some input values for providing a service, and these input values will partially determine the results of the Web service. One may think that a customer goal should not only describe the results required, but also what information the customer has available and is willing to disclose for achieving such results, as it plays a role in the discovery process. However, the input required by available Web services, some of which might solve the customer's goal, is not known by the customer before-hand, and it can greatly vary among them. For example, any Web service might require authentication (user name), as WS4 does, and a customer defining e.g. goal G3 cannot anticipate whether his user details will be necessary for achieving his goal. In the best case the customer can only *guess*, when defining his goal, what input values will be required by Web services satisfying such goal.

In addition, the customer might have a considerable volume of information from which input values to Web services can be obtained. As an example, consider a business process that acts as a customer and that has access to information in corporate databases: input values for a Web service can be extracted from these databases if this is required for achieving a goal of the process. Finally, part of this information might be sensitive and, therefore, it should not be disclosed to third parties before establishing some trust relation. For example, and even though a customer trying to achieve G3 might be willing to disclose his credit card details for purchasing shares of an investment fund, he will probably not do so before the specific Web service that can be used for this purpose has been identified and a trust relation has been established with it.

For these reasons, we will assume that the customer will keep his information locally i.e. it will not be attached to the goal, so that the description of expected results can be disclosed to third parties such as Web service registries without disclosing customer's information. In particular, we will assume that a requester knowledge base KB_c, containing all the information known by the customer and usable for achieving the goal, exists and it is kept locally. If a human user participates in the discovery process, he can extend KB_c with additional information required by some Web service and not in KB_c.

Additionally, there might be potential input information contained in the definition of a goal. For example, G3 gives details of the customer credit card and of the fund the requester wants to purchase. These details might be later used as input to a Web service for achieving the goal and, therefore, they will be considered as potential input values and included in KB_c. Furthermore, if this kind of information included in the goal has a sensitive nature, as it is the case of the credit card details of the customer in G3, it will be replaced by a restriction derived from its definition but which does not reveal sensitive details e.g. credit card number and expiration date. For example, G3 will only say that a Visa credit card (assuming the customer credit card is a Visa) has to be used for the purchase, without giving further details. The result of eliminating sensitive information from the formalization of G3 is:

$$\forall x.res_{G3}(x) \leftrightarrow \exists cc.purchase(x) \wedge ofItem(x, myFund) \wedge withPaymentMethod(x, cc) \wedge visaCreditCard(cc)$$

Notice that the pre-processing of goals and the identification of what information the customer is *willing to* disclose for achieving a given goal is currently done manually. The automation of goal pre-processing and the identification of information the customer is willing to disclose from KB_c can be based in the definition of information disclosure policies (Olmedilla *et al.* 2004). The use of this type of policies will be part of our future work.

Web service capability descriptions

A Web service $\mathcal{W}$ can be seen as a function that maps certain input values and the current state of the world to a service $serv_i \in \mathcal{A}^{\mathcal{W}}$, with certain outputs and effects.

However, we will drop the dependency of the Web service results on the current state. The reason is that the discovery process will generally have partial visibility on the current state of the world and, thus, a distributed evaluation of capability descriptions will be required. This means that evaluating the dependency between the current state of the

world and the results of a Web service might require knowing certain information that might only be accessible via other Web services, which would have to be in turn discovered and executed during the discovery process. This can considerably degrade efficiency. For example, a precondition of WS4 might say that only investment funds currently commercialized can be purchased. For evaluating this precondition and, thus, for determining whether the purchase of a particular fund can be requested to this Web service, we will most likely need to locate an invoke a Web service that can tell us what funds are currently commercialized.

Given this simplification, we can formalize the function offered by a Web service in first-order logic through a predicate $res_{\mathcal{W}}(x, i_1, \ldots, i_n)$ defined as follows:

$$\forall x, i_1, \ldots, i_n . res_{\mathcal{W}}(x, i_1, \ldots, i_n) \leftrightarrow in_{\mathcal{W}}(i_1, \ldots, i_n) \wedge \phi_{\mathcal{W}}(x, i_1, \ldots, i_n) \quad (1)$$

where x denotes a result of the Web service (either an output or an effect) and $i_1, \ldots, i_n$ denote the input values given by the customer. Intuitively, $res_{\mathcal{W}}(x, i_1, \ldots, i_n)$ formalizes the relation between the results of the service provided by $\mathcal{W}$ and the input values given by the customer. From the predicates defining this relation, $in_{\mathcal{W}}(i_1, \ldots, i_n)$ captures the conditions input values must fulfill (Web service inputs), and $\phi_{\mathcal{W}}(x, i_1, \ldots, i_n)$ denotes a first-order formula over variables $x, i_1, \ldots, i_n$ that defines the relation between Web service results (x) and valid input values. Notice that, for simplicity, we do not differentiate between outputs and effects.

For example, $res_{WS1}(x, co)$ would be defined in terms of the following predicates:

$\forall co . in_{WS1}(co) \leftrightarrow country(co) \wedge inContinent(co, europe)$

$\forall x . co . \phi_{WS1}(x, co) \leftrightarrow \exists f, inst . infoProvision(x) \wedge ofItem(x, f) \wedge fund(f) \wedge commercializedBy(f, inst) \wedge institution(inst) \wedge fromCountry(inst, co)$

Input-independent descriptions. A predicate of the type introduced above formalizes the relation between input values and results of the Web service. However, a characterization of the results of the Web service, dropping the relation between these and input values, can be of great interest for increasing the efficiency of the discovery process, as we will see in the next section. Therefore, we will introduce an alternate description of the Web service functional value that does not take into account the afore-mentioned relation. In particular, we will define a predicate $res'_{\mathcal{W}}(x)$ as follows:

$\forall x . res'_{\mathcal{W}}(x) \leftrightarrow \phi'_{\mathcal{W}}(x)$

where $\phi'_{\mathcal{W}}(x)$ is a first-order formula that formalizes the conditions results of the Web service fulfill. For WS1, this predicate would be defined as:

$\forall x . res'_{WS1}(x) \leftrightarrow \exists f, inst, co . infoProvision(x) \wedge ofItem(x, f) \wedge fund(f) \wedge commercializedBy(f, inst) \wedge institution(inst) \wedge fromCountry(inst, co) \wedge country(co) \wedge inContinent(co, europe)$

As it can be seen, this type of description is symmetric to the description of goals.

Relation between descriptions The relation between predicate $res'_{\mathcal{W}}(x)$ and predicate $res_{\mathcal{W}}(x, i_1, \ldots, i_n)$ describing the functional value of a Web service $\mathcal{W}$ can be formalized as follows:

$\forall x . res'_{\mathcal{W}}(x) \leftrightarrow \exists i_1, \ldots, i_n . res_{\mathcal{W}}(x, i_1, \ldots, i_n)$

Intuitively, the formula above states that given any result x for which $res'_{\mathcal{W}}(x)$ is true, there will be some input values for which $res_{\mathcal{W}}(x, i_1, \ldots, i_n, w)$ is also true i.e. both predicates describe the same results as possible outcomes of the use of Web service $\mathcal{W}$, but considering or not the dependency on input values. In a nutshell, the input-independent description provides a simplified, but useful, view on the functional value of the Web service. We conjecture that the input-independent view can be automatically extracted from the input-dependent view. However, this is subject of future work.

Publication of descriptions. When a Web service is published so that it can be used by potential customers, the provider will also publish the formal description (both input-dependent and input-independent) of the Web service functional offer at a (not necessarily centralized) registry. The Web service discovery process will have to access this registry in order to retrieve these descriptions.

However, and although the realization of the discovery process that will be proposed in the next section is based on the first-order input-dependent and input-independent types of descriptions introduced above, it will introduce some changes to the descriptions expected. As it will be seen in the next section, these changes are: a) the expressivity allowed for describing the input-independent capability of a Web service will be restricted to a certain Description Logic, and therefore the description used will be a Description Logics (DL) (Nardi *et al.* 2003) concept definition, and b) the input-dependent capability will be described by a set of pairs (Logic Programming -LP- rule (Lloyd 1987), DL concept definition), as it will be explained in the next section. The descriptions that will be actually published to the registry are these ones.

Domain model.

The formal description of goals and Web service capabilities is done in terms of an underlying domain model, which will be also formally defined. In particular, the domain model will be given by a set $\mathcal{O}$ of ontologies. In our use case, this set includes: a) an ontology Geo of geographic locations, b) an ontology Pay of payment methods, c) an ontology Inv of investment funds, and d) an ontology Res of types of results of a service provision e.g. information provision, purchase, booking, etc. Each ontology defines classes (unary predicates), relations (n-ary predicates), individuals (constants), and general axioms (e.g. subclass axioms) that together provide the domain vocabulary for its particular domain.

For simplicity, we will assume that the domain model used for describing goals and capabilities is the same. In a practical setting, this will usually not be the case, as different parties might use different formalizations of the same domain, and a mediation mechanism will be necessary.

Expressivity. We introduce some restrictions on the expressivity of the formal language used for formalizing domain models. In particular, we restrict it to Description Logic Programs (DLP) (Grosof *et al.* 2003), the language underlying WSML-Core (de Bruijn, J. (ed.) 2005). The reason is that: a) proper language extensions of DLP (Kifer *et al.* 2005) in the direction of Logic Programming, properly semantically layered on top of DLP, are possible e.g. WSML-Flight (de Bruijn, J. (ed.) 2005) and b) proper language extensions of DLP in the direction of First-Order Logic, properly semantically layered on top of DLP, are also possible e.g. WSML-DL (de Bruijn, J. (ed.) 2005)[2]. This will be an important feature for our Web service discovery process, as LP and DL extensions of DLP will be used at the different phases of discovery introduced by our model.

By restricting domain ontologies to a common fragment of LP and DL, we will be able to use *the same* domain ontologies when extensions of DLP in the direction of DL are used, as well as when extensions of DLP in the direction of LP are used i.e. we do not need to duplicate domain ontologies to be used under these two different paradigms. Remarkably, and according to (Volz 2004), it turns out that *most* OWL (and DAML+OIL) ontologies in popular ontology libraries fall in the fragment of First-Order Logic captured by DLP.

Restrictions in the expressivity allowed for describing goals and Web services will be also introduced, but they will be described in the next section.

Web service discovery process

The formal description of requester goals and Web service capabilities provides the ingredients for (semi)automating the Web service discovery process, as automated reasoning can be applied to determine whether a certain relation holds between the set of results required by a given goal and the set of results offered by available Web services. In particular, we want to check whether the results offered by a given Web service for some input values are a) a superset of, b) a subset of, c) a set equivalent to, or d) a set intersecting with the set of results required by the customer. That means that, given the descriptions of a goal $\mathcal{G}$ and a Web service $\mathcal{W}$, both referring to the same domain model $\mathcal{O}$, we want to check whether given $\mathcal{O}$ and the knowledge base KB_c some of the following relations hold:

$$\mathcal{O}, KB_c \models \forall x.(\exists i_1, \ldots, i_n.res_{\mathcal{G}}(x)\mathcal{R}res_{\mathcal{W}}(x, i_1, \ldots, i_n)) \quad (2)$$

$$\mathcal{O}, KB_c \models \exists x, i_1, \ldots, i_n.res_{\mathcal{G}}(x) \wedge res_{\mathcal{W}}(x, i_1, \ldots, i_n) \quad (3)$$

with $\mathcal{R} \in \{\leftarrow, \leftrightarrow, \rightarrow\}$. Notice that the expressions above allow the use of the same Web service multiple times with different input values, for obtaining all the results required, but not the composition of different Web services.

In the following, we present how the Web service discovery process is articulated in two phases in order to locate Web services that, for certain input values the customer knows and is willing to disclose to the Web service, provide all or some of the results required by a given goal.

1st phase: results-based Web service discovery

Let us consider G3 and WS4 and let us suppose we want to evaluate whether WS4 can offer, for some input values, all the results requested by G3. In this setting, we would have to evaluate the following expression (see (2))

$$\mathcal{O}, KB_c \models \forall x.(\exists fund, cc, user.res_{G3}(x) \rightarrow res_{WS4}(x, fund, cc, user))$$

If we use classical first-order semantics (Fitting 1996), the Open World Assumption (OWA) is made when trying to prove the expression above. However, input values, which will have to be latter submitted to the Web service in order to get appropriate results for fulfilling the goal, can only come from information the customer using the Web service knows. We believe that, in practice, values for variables $fund$, cc and $user$ can only come from the domain knowledge given by $\mathcal{O}$ and the knowledge base KB_c and, therefore, we want to locally close the world (Heflin & Munoz-Avila 2002) to obtain possible assignments for these variables i.e. we want to consider only values from the knowledge base formed by the union of $\mathcal{O}$ and KB_c.

For evaluating the expression above locally closing the world as explained in the previous paragraph, access not only to domain knowledge (domain ontologies $\mathcal{O}$) but also to the customer's knowledge base KB_c is required. As the customer is expected to keep his information i.e. KB_c locally, we will only be able to have access to the full knowledge from which input values can be obtained and, therefore, to evaluate the expression above, on the customer side.

In this setting, we first need to retrieve the capability descriptions of available Web services for their evaluation on the customer side, as this kind of descriptions will be usually stored in a registry or set of registries external to the customer. However, we do not want to retrieve *all* available Web services, as there might exist a big number of them. Instead, we want to retrieve only *relevant* Web services for their evaluation on the requester side, so that the number of Web services that must be evaluated is reduced. This is precisely the purpose of the results-based discovery phase: efficiently retrieving Web services relevant for the goal at hand in terms of the complete set of results they offer, therefore reducing the number of Web services to be evaluated in more detail (considering possible input values and the relation between these and the results of the Web service).

Kind of descriptions used. In this phase we will make use of input-independent descriptions of Web service capabilities, as we will not have access to the customer knowledge base and, therefore, to all possible input values. Therefore, we will use predicates $res_{\mathcal{G}}(x)$ and $res_{\mathcal{W}}(x)$, which formally describe the set of results expected by the customer and the set of results offered by the Web service.

Relations. The relations that are of interest are the set-theoretic relations between the sets of results required by the goal and offered by a Web service. In particular, we have to evaluate, given goal $\mathcal{G}$ and a Web service $\mathcal{W}$ whether:

[2]This layering is argued to be incorrect in (Horrocks *et al.* 2005). However, we recommend reading (Kifer *et al.* 2005) for an answer to the concerns raised and a clarification of the issue.

$\mathcal{O} \models \forall x.res_{\mathcal{G}}(x) \mathcal{R} res_{\mathcal{W}}(x)$

or

$\mathcal{O} \models \exists x.res_{\mathcal{G}}(x) \wedge res_{\mathcal{W}}(x)$

holds, with $\mathcal{R} \in \{\leftarrow, \leftrightarrow, \rightarrow\}$. In general, and given a goal $\mathcal{G}$, we will want to obtain Web services $\mathcal{W}$ for which some of the above expressions hold i.e. Web services that offer at least some results from the set requested. For a detailed discussion on the possible ranking of Web services according to the relation between the goal and the Web service, also considering meta-annotations denoting different intentions, we refer the reader to (Keller *et al.* 2005).

Restrictions on expressivity. Evaluating, for a given goal, the individual relation between this goal and each available Web service can be rather costly. In order to improve the efficiency of this process, so that Web services providing results required by the customer can be quickly identified and returned to the customer for further evaluation, we will restrict the expressivity allowed for defining predicates $res_{\mathcal{G}}(x)$ and $res_{\mathcal{W}}(x)$ to a subset of full first-order logic, namely to the $\mathcal{SHOIN}(D)$ Description Logic (Nardi *et al.* 2003), which is the Description Logic underlying OWL-DL (Horrocks & Patel-Schneider 2003) and WSML-DL[3] (de Bruijn, J. (ed.) 2005).

The reason for using this subset of first-order logic is that the unary predicates $res_{\mathcal{G}}(x)$ and $res_{\mathcal{W}}(x)$ can be regarded as the definition of Description Logics concepts, and the set-theoretic relation between the formalized sets of results can be determined through DL subsumption reasoning (Nardi *et al.* 2003) over these concepts, for which efficient reasoners such as RacerPro (RAC 2005) exist, and for which off-line classification is possible, as we will see later.

For example, if we consider goal G1, predicate $res_{G1}(x)$ defined as:

$\forall x.res_{G1}(x) \leftrightarrow \exists f, inst.infoProvision(x) \wedge ofItem(x, f) \wedge fund(f) \wedge commercializedBy(f, inst) \wedge institution(inst) \wedge fromCountry(inst, spain)$

can be seen as equivalent to the definition of a concept RG1 (using DL syntax (Nardi *et al.* 2003)) as follows:

$RG1 \equiv infoProvision \sqcap \exists ofItem.(fund \sqcap \exists commercializedBy.(institution \sqcap \exists fromCountry.\{spain\}))$

Web service classification and querying. In order to avoid individually checking the subsumption relation between a given goal and the capability of each Web service available, which would result in unacceptable response times if a relatively big number of Web services has to be evaluated, we exploit the possibility of classifying Web services off-line at publication time (Li & Horrocks 2003) .

For this purpose, each time a new Web service is published the concept formalizing the set of results it offers is added to the T-Box of a DL reasoner (Nardi *et al.* 2003) (possibly integrated with registries), and this T-Box is classiffied. Given a particular goal $\mathcal{G}$ and the DL concept RG defining the set of results required, we can query the T-Box for Web services whose set of results i.e. the concept defining the results they offer: a) is equivalent to, b) is more general than, c) is more specific than, and d) has a non-empty intersection[4] with RG. In this way, we can obtain all relevant Web services for the goal at hand with only four queries. The key benefit of following this approach is that once the concepts describing the results of available Web services are classified, the response times for incoming queries are quite low, enabling an efficient retrieval of relevant Web services.

Evaluation. For evaluation purposes, we have used the RacerPro (RAC 2005) DL reasoner[5]. First, we have formalized the goals and Web services from our use case and checked their subsumption relation by submitting the appropriate queries to the DL reasoner. The results are summarized as follows:

- G1: RWS2 is equivalent to RG1. RWS1 and RWS3 are more general (subsume) RG1.
- G2: RWS3 subsumes RG2. RWS2 is subsumed by RG2. RWS1 has a non-empty intersection with RG2.
- G3: RWS5 subsumes RG3 (assuming $myFund$ is a European fund).

If the concept defining the results of a Web service subsumes the concept defining the results required by the goal, that means that the Web service can offer all the results required plus additional results not required. If the subsumption relation is in the other direction, that means that the Web service only offers a subset of the results required. If they are equivalent, that means that the set of results offered and requested coincide. Finally, if they only intersect, that means that the Web service offers some results from the set requested, and some other results not requested. As it can be seen, we obtain for each goal the Web services that are relevant for the goal in the sense that they provide some or all the results required by the customer.

The relations considered between the concepts defining the complete results offered by a Web service and the results expected by the customer have been used by several works e.g. (Keller *et al.* 2005; 2006; Li & Horrocks 2003; Paolucci *et al.* 2002; Benatallah *et al.* 2005) and they partly have their roots in the work on automated software component retrieval (Zaremski & Wing 1997). Meta-annotations of concepts in order to capture the intention of the modeler when defining their sets of required or offered results are also considered in

[3] To be precise, the Description Logic underlying WSML-DL is $\mathcal{SHIQ}(D)$, but future versions of the language might allow for the use of nominals.

[4] In practice, as RacerPro does not allow for this type of query, we actually query for Web services not intersecting the goal and we return Web services not in this set and not having a subsumption relation (equivalent, more general, more specific) with the goal.

[5] As RacerPro does not fully support nominals, we translate nominals into pair-wise disjoint atomic concepts. As presented in (Horrocks & Sattler 2002), incorrect inferences can be drawn for A-Box reasoning applying such translation, but the the subsumption and satisfiability relations computed will be correct, which is enough for our purposes.

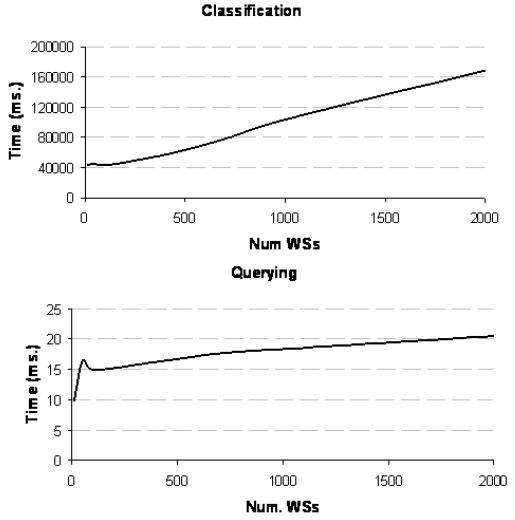

Figure 1: Classification and querying response times

(Keller *et al.* 2005), as well as how they affect the matching relation between goals and Web services.

The second part of our evaluation has focused on the response times of T-Box classification and querying when a big number of Web services is available. For this purpose, we have generated random variations of the Web services from our use case and measured these response times. The results are shown in Figure 1. The X axis shows the number of Web services available, and the Y axis the time in milliseconds. As it can be seen, classification is time-consuming. However, this is not a problem as it can be done off-line i.e. at publication time and without affecting the response times of the discovery process. Once Web service capabilities are classified, the response time for an incoming query (goal) is around 20 milliseconds for 2000 Web services available, as it can be observed from the figure.

2nd phase: considering input values

Once we have identified relevant Web services in terms of the results they offer, their complete descriptions, including the input-dependent description of their capabilities, will be retrieved and available to the customer. We can now evaluate on the customer side, where KB_c is accessible, the expressions (2) and (3) above for each Web service retrieved, locally closing the world for querying possible values to be assigned to input variables i.e. we can evaluate whether appropriate input values can be provided from $\mathcal{O}$ and KB_c to each such Web service, and whether the set of results offered for such input values, which will be in general a subset of the complete set of results offered by the Web service, satisfies the goal of the customer. Notice that input values will not be actually provided to Web services during discovery, but the evaluation of whether appropriate input values are available for relevant Web services and how they restrict the results offered will be a formal one, only based on the static description of goals and Web service capabilities and on the information in $\mathcal{O}$ and KB_c, without any direct communication with external parties.

However, expressions (2) and (3) above cannot be evaluated using pure first-order theorem proving if we want to locally close the world for obtaining input values. For this reason, we propose a work-around in three steps, based on query answering for Logic Programs and DL subsumption reasoning.

Step 1: querying for input values. First, we want to retrieve from $\mathcal{O}$ and KB_c, and for each Web service, possible sets of input values that satisfy the requirements of the Web service considered. For this task, we want to use LP query answering, as the Closed-World Assumption (CWA) must be made. Furthermore, the size of the knowledge bases from which we will have to retrieve appropriate input values might be big, and LP reasoners are highly optimized for this task.

We remind from the previous section that predicate $res_{\mathcal{W}}(x, i_1, \ldots, i_n)$, which formalizes the results of a Web service in relation to the input values provided, was defined using a predicate $in_{\mathcal{W}}(i_1, \ldots, i_n)$ that formalized the conditions such input values must fulfill (see (1)). Therefore, only values for variables $i_1, \ldots, i_n$ satisfying this predicate are valid input values which can make expressions (2) and (3) true.

In this context, we can translate the conditions on input values formalized by $in_{\mathcal{W}}(i_1, \ldots, i_n)$ into an LP query, in order to retrieve only valid input values for the Web service. In particular, predicate $in_{\mathcal{W}}(i_1, \ldots, i_n)$ will be translated into the body of an LP rule. Let us take WS1 as an example, for which $in_{WS1}(co)$ was defined as follows:

$$\forall co.in_{WS1}(co) \leftrightarrow country(co) \wedge inContinent(co, europe)$$

The rule resulting from the translation (written using FLORA-2[6] syntax) is:

$$\begin{aligned} \_\#(Input) : assignmentSet[forCapability \rightarrow RWS1P, \\ assg \rightarrow \_\#1(Input), fromInputs \rightarrow Input], \\ \_\#1(Input) : assignment[to \rightarrow \text{"!}co\text{"}, value \rightarrow Input] \\ :- Input : country[inContinent \rightarrow europe]. \end{aligned}$$

Notice that the body of the rule, to the right of symbol $:-$, corresponds to the LP translation of the first-order predicate $in_{WS1}(co)$, querying for facts $Input$ that are an instance of $country$ and that are located $inContinent\ europe$[7].

If we evaluate this rule using an LP reasoner e.g. FLORA-2 over the knowledge base formed by $\mathcal{O}$ and KB_c, the body of the rule will query for input values satisfying the conditions described. For each set of input values satisfying the conditions posed by such body, an (anonymous) instance of the class $assignmentSet$, representing a set of assignments

[6] http://flora.sourceforge.net

[7] This translation is currently done manually. Automating this translation is subject of future work.

of values to some parameters, will be generated. This instance will have the following information: for what capability this set of assignments was generated, the set of assignments itself (of some values retrieved by the query in the body to some parameters of the capability -!co-), and what input values generated it. What kind of capabilities and parameters are used will be explained in the next step.

For example, if in the knowledge base considered (formed by $\mathcal{O}$ and KB_c) there is an instance $spain$ that is defined as being a country located in continent $europe$, the example rule above will yield the facts _#(spain) and _#1(spain) as follows:

$$\begin{aligned} \_\#(spain) : assignmentSet[forCapability \rightarrow RWS1P, \\ assg \rightarrow \_\#1(spain), fromInputs \rightarrow spain]. \\ \_\#1(spain) : assignment[to \rightarrow \text{"!}co\text{"}, value \rightarrow spain]. \end{aligned}$$

Step 2: parameterizing sets of results. The set of results offered by a Web service will depend on the input values given to it. In this step we will introduce DL concepts describing the set of results offered by the Web service, but depending on some parameters. These parameters will be replaced by input values obtained in the previous step.

Remind from the previous section that, given a Web service $\mathcal{W}$, predicate $res_{\mathcal{W}}(x, i_1, \ldots, i_n)$ was defined using also a predicate $\phi_{\mathcal{W}}(x, i_1, \ldots, i_n)$ that formalized the relation between results and input values. We will capture this relation by defining a DL concept that characterizes the set of results provided by the Web service, but with certain parameters. Let us illustrate this point by translating predicate $\phi_{WS1}(x, co)$:

$$\forall x.co.\phi_{WS1}(x, co) \leftrightarrow \exists f, inst.infoProvision(x) \wedge ofItem(x, f) \wedge fund(f) \wedge commercializedBy(f, inst) \wedge institution(inst) \wedge fromCountry(inst, co)$$

into the DL concept RWS1P[8]:

$$RWS1P \equiv infoProvision \sqcap \exists ofItem.(fund \sqcap \exists commercializedBy.(institution \sqcap \exists fromCountry.!co))$$

where $!co$ denotes a parameter that must be replaced by an actual value. The values that will replace $!co$ will be obtained by the LP rule introduced before, yielding a new concept $RWS1P'$. For example, $!co$ would be replaced by $spain$ if the facts derived from the evaluation of the rule are the ones shown above i.e. we will have:

$$RWS1P' \equiv infoProvision \sqcap \exists ofItem.(fund \sqcap \exists commercializedBy.(institution \sqcap \exists fromCountry.spain))$$

In a nutshell, we provide parameterized formalizations of the set of results a Web service can provide. These parameters will correspond to those input values, required by the Web service, that restrict in some way the set of results provided. The rules extracted in the previous step will obtain valid input values for the Web service *from the set $\mathcal{O}$ of domain ontologies and from the knowledge base KB_c i.e. locally closing the world*, which will be used to replace the parameters in the description of the set of results offered by the Web service. This way, the set of results offered by the Web service for available input values is obtained.

Notice that multiple assignments for parameters in the DL concept definition, corresponding to different valid sets of input values, might be generated by the LP rules. We will perform the parameter replacement in the DL descriptions for each such set of input values, yielding concepts $RWSP_i$ for each such replacement. The set of results that can be provided by the Web service when executing with all these sets of input values will be defined by a concept $RWSP' \equiv RWSP_i \sqcup \ldots \sqcup RWSP_n$ i.e. the set of results offered is the union of the sets of results obtained for the possible execution with each of these sets of input values.

Step 3: using DL subsumption reasoning. Once we have applied the LP rules over $\mathcal{O}$ and KB_c, obtaining the DL concept $RWSP'$ defining the results offered by a given Web service for the input values that can be provided by the requester, we can again apply DL subsumption reasoning for determining the relation between this concept and the concept defining the results expected by the customer. This way, we will obtain the relation between the results required and the results the relevant Web services retrieved in the first phase can provide for the input values available. The relations that are of interest are the same ones introduced in the first phase of the discovery process, which actually correspond to the relations considered by expressions (2) and (3) for available input values.

In a nutshell, we will determine whether some Web services can provide some or all the results required for achieving the customer's goal given the input values that can be provided by the customer. Notice that, in order to be able to efficiently check the relation between the customer's goal and the set of results offered by the Web service for available input values, the expressivity of the DL concept formalizing such set of results is restricted to $\mathcal{SHOIN}(\mathcal{D})$, like in the first phase of our model.

Evaluation. For evaluation purposes we have used FLORA-2 as LP reasoner, and again RacerPro as DL reasoner. For the goals in our use case, we obtained the following results (assuming $\mathcal{O}$ defines the country $spain$, and KB_c contains the user information $myUser$; the funds category FIAMMCNMV, the credit card $myCC$, and the fund $myFund$ are extracted from the description of the goals where they are used):

- G1: the set of results RWS1P', parameterized with input $spain$, is equivalent to RG1. There are no input values (institution and fund category) available for WS2 and WS3 and, therefore, they cannot be used to achieve the goal.
- G2: the set of results RWS3P', parameterized with input FIAMMCNMV, is equivalent to RG2. RWS1P', parameterized with input $spain$, intersects RG2. Input for WS2 (institution) is not available.
- G3: the set of results RWS4P', parameterized with input $myFund$ and $myCC$, and given that $myUser$ is a valid

[8]This translation is currently done manually, and its automation will be studied in the future.

user for WS4, is equivalent to RG3.

Therefore, and given $\mathcal{O}$ and KB_c above, WS1 can provide all the results required for achieving G1 and part of the results for achieving G2, WS3 can provide all the results required for achieving G2, and WS4 can provide all the results required for achieving G3.

Interoperability. We use LP rules in combination with Description Logic concepts, which can lead to interoperability problems as LP semantics are slightly different from classical first-order semantics. First, in order to avoid duplicating the definition of ontologies, under LP semantics and under first-order semantics, and possible unintended semantic differences in models describing the same domain, we need to provide formalizations of domain models that can be consistently used by LP extensions of DLP and by first-order extensions of DLP. This is achieved by restricting the expressivity of domain models (and, therefore, of the customer knowledge bases expressed in terms of this model) to DLP, as the extensions in the direction of LP and in the direction of FOL used in our approach are properly semantically layered on top of DLP and, thus, all consequences inferred from a DLP specification are also valid consequences in the LP and FOL extensions used.

Second, Description Logics usually make the Open-World Assumption (OWA), while the Closed-World Assumption (CWA) is made in Logic Programming, which can constitute a difficulty for the combined used of both languages, as the consequences entailed can vary depending on the kind of assumption made. However, in our setting we precisely exploit this feature of each paradigm. LP is used for querying input values from a defined (closed) knowledge base and, thus, the CWA must actually be made. On the contrary, for reasoning with the DL concepts describing the set of results requested and offered, the OWA must be made, as the details of results not specified should not be deemed to be false.

In a nutshell, we use an LP reasoner for obtaining valid input values, locally closing the world to $\mathcal{O}$ and KB_c when querying for such input values. The interface between LP reasoning and DL reasoning is the assignment of the input values obtained to the parameters in DL concept definitions. After replacing these parameters by the values obtained, FOL semantics are used and the OWA is made in the DL subsumption reasoning performed. Therefore, we exploit the features of each type of semantics and reasoning at different points in time.

Discussion

In this paper, we have presented a Web service discovery process articulated in two phases in order to be able to consider the input values required by Web services and how these restrict the results of the Web service, but keeping the process efficient and without requiring customers to anticipate what information might be needed as input values for Web services satisfying their goal and to include this information along with their goals.

We have introduced restrictions on the expressivity allowed for semantic descriptions so that these features are possible, and we have realized the process relying on the joint use of Description Logics subsumption reasoning and query answering for logic programs. We have also implemented a preliminary prototype for carrying out the process presented in the paper, which will be further evolved.

Future work will concentrate on further analyzing possible interoperability issues that might arise in the combined use of DL and LP, on integrating the descriptions necessary for our proposal in existing frameworks, and on automating certain parts of the process which are currently done manually.

Related work. The major advantages introduced by our proposal with respect to previous works in the area are the following:

a) We are able to consider the relation between the input values provided to a Web service and its results during the discovery process, which is not possible in approaches that exclusively rely on Description Logics e.g. (Li & Horrocks 2003; Paolucci *et al.* 2002; Benatallah *et al.* 2005; Gonzalez-Castillo, Trastour, & Bartolini 2001; Verma *et al.* 2004; Noia *et al.* 2003).

Being able to formalize and consider the relation between available input values and the set of results offered by a Web service is beyond what can be achieved by using pure Description Logic approaches such as (Benatallah *et al.* 2005). This increases the complexity of the descriptions required and of the discovery process. However, we believe it has the following benefits: i) the results yielded by the Web service discovery process are more accurate, as they are obtained based on a description of the functional offer of Web services closer to the real function they implement, and ii) not only what Web services match a given goal and to what extent can be obtained, but also for what input values such matches are expected; this can serve to guide the actual interaction with the Web service in order to request the provision of the desired service. Still, our model allows for three (increasingly complex) types of discovery: discovery based only on the results offered by available Web services, evaluation of whether the customer has valid input values for relevant Web services, and evaluation of whether valid input values restrict the results of relevant Web services in a way such that these Web services still satisfy the goal.

b) We can efficiently identify relevant Web services for further evaluation even when a big number of Web services is available, which is usually not possible when pure LP reasoning is used e.g. (Kifer *et al.* 2004). Furthermore, following a pure LP-based approach implies making the Closed-World Assumption in all proof obligations, which presents certain problems if Web service capability descriptions are not completely accurate and if no communication with Web services is allowed.

c) We do not expect customers to describe the information they can provide for fulfilling their goals along with their goal definition, but we expect them to keep it locally without requiring its explicit listing with each of their goals; we believe this is a more realistic assumption than which is implicitly made by most DL-based works, for the reasons introduced in this paper.

Communication with Web services. The approach presented exclusively relies on the formal and static description of the function offered by Web services, and no direct communication with Web services will take place before the discovery process ends. The reason is that direct communication with Web services is costly, and it should not be done before the number of Web services that can potentially fulfill a goal is considerably narrowed down. However, the formal descriptions used in the discovery process will be a static, public characterizations of the Web service functional offer, being complete but not correct (Preist 2004; Lara & Olmedilla 2005). Thus, for a complete guarantee that Web services selected during the discovery process will actually fulfill the customer's goal, direct interaction with them will be required after discovery in what can be called service definition or contracting phase (Preist 2004; Keller *et al.* 2005; Lara & Olmedilla 2005).

Note on the language used. For evaluating the model presented we have used WSML, restricting domain models and the customer knowledge base to WSML-Core (as DLP is the formalism underlying this language and it provides an interesting interoperability layer), the DL definition of concepts to WSML-DL with nominals, and the definition of LP rules to WSML-Flight. WSML-DL and WSML-Flight are properly layered on top of WSML-Core and, therefore, every WSML-Core expression is a valid WSML-DL and WSML-Flight expression. The choice of the WSML family of languages is justified by the fact that it provides the layering of languages we require for realizing our discovery model.

Acknowledgements.

This work has been partially funded by CDTI under grant CDTI05-0436. The author also thanks Holger Lausen for fruitful discussions, and Jos de Bruijn and Axel Polleres for comments on a preliminary version of this paper.

References

Benatallah, B.; Hacid, M.; Leger, A.; Rey, C.; and Toumani, F. 2005. On automating web services discovery. *VLDB* 14:84–96.

de Bruijn, J. (ed.). 2005. The Web Service Modeling Language WSML. WSML d16.1v0.21.

Fitting, M. 1996. *First order logic and automated theorem proving*. Springer Verlag, 2nd edition.

Gonzalez-Castillo, J.; Trastour, D.; and Bartolini, C. 2001. Description logics for matchmaking of services. In *KI-2001 Workshop on Applications of Description Logics*.

Grosof, B.; Horrocks, I.; Volz, R.; and Decker, S. 2003. Description logic programs: Combining logic programs with description logic. In *12th International Conference on the World Wide Web (WWW-2003)*.

Heflin, J., and Munoz-Avila, H. 2002. LCW-based agent planning for the semantic web. In *AAAI Workshop on Ontologies and the Semantic Web*.

Horrocks, I., and Patel-Schneider, P. F. 2003. Reducing OWL entailment to Description Logic satisfiability. In *ISWC 2003*.

Horrocks, I., and Sattler, U. 2002. Optimised Reasoning for $\mathcal{SHIQ}$. In *ECAI 2002*, 277–281.

Horrocks, I.; Parsia, B.; Patel-Schneider, P.; and Hendler, J. 2005. Semantic web architecture: stack or two towers? In *Principles and Practice of Semantic Web Reasoning*.

Keller, U.; Lara, R.; Lausen, H.; Polleres, A.; and Fensel, D. 2005. Automatic location of services. In *ESWC 2005*.

Keller, U.; Lara, R.; Lausen, H.; and Fensel, D. 2006. *Semantic Web Services: Theory, Tools and Applications*. chapter Semantic Web Service Discovery in the WSMO Framework. To appear.

Kifer, M.; Lara, R.; Polleres, A.; Zhao, C.; Keller, U.; Lausen, H.; and Fensel, D. 2004. A Logical Framework for Web Service Discovery. In *Semantic Web Services Worshop at ISWC 2004*.

Kifer, M.; de Bruijn, J.; Boley, H.; and Fensel, D. 2005. A realistic architecture for the semantic web. In *RuleML-2005*.

Lara, R., and Olmedilla, D. 2005. Discovery and Contracting of Semantic Web Services. In *W3C Workshop on Frameworks for Semantics in Web Services*.

Li, L., and Horrocks, I. 2003. A Software Framework for Matchmaking Based on Semantic Web Technology. In *WWW'03*.

Lloyd, J. W. 1987. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag.

Nardi, D.; Baader, F.; Calvanese, D.; McGuinness, D. L.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook*. Cambridge.

Noia, T. D.; Sciascio, E. D.; Donini, F. M.; and Mongiello, M. 2003. A system for principled matchmaking in an electronic marketplace. In *Proc. Intl. Conf. on the World Wide Web 2003 (WWW2003)*.

Olmedilla, D.; lara, R.; Polleres, A.; and Lausen, H. 2004. Trust Negotiation for Semantic Web Services. In *SWSWPC Workshop at ICWS 2004*.

Paolucci, M.; Kawamura, T.; Payne, T.; and Sycara, K. 2002. Semantic Matching of Web Service Capabilities. In *ISWC*, 333–347. Springer Verlag.

Preist, C. 2004. A Conceptual Architecture for Semantic Web Services. In *ISWC 2004*.

2005. RacerPro user's guide. version 1.8. Technical report, RACER Systems GmbH & Co. KG.

Verma, K.; Sivashanmugam, K.; Sheth, A.; and Patil, A. 2004. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*.

Volz, R. 2004. *Web Ontology Reasoning with Logic Databases*. Ph.D. Dissertation, AIFB, Karlsruhe.

Zaremski, A., and Wing, J. 1997. Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 6:333–369.