A flexible model for Web service discovery

Rubén Lara[1], Miguel Ángel Corella[2] and Pablo Castells[2]

[1] Tecnología, Información y Finanzas, Madrid, Spain
rlara@afi.es
[2] Universidad Autónoma de Madrid, Spain.
{miguel.corella,pablo.castells}@uam.es

Abstract. The advent of the SOA paradigm is expected to cause an increase in the number of available Web services. In this setting, advanced facilities for the discovery of Web services that provide a given functionality are required so that this increase does not create a bottleneck for the exploitation of services in an SOA. In this paper, we present a model for the discovery of Web services which relies on alternative views of Web service functional capabilities, allowing for different trade-offs between the accuracy of discovery results and the efficiency of the discovery process. Up to three filtering steps, with increasing complexity, can be successively applied in order to refine discovery results. Furthermore, the use of taxonomies of functional categories, close to the intuition of average users, is introduced by our model with a two-fold purpose: a) supporting the user in the description of goals and Web services, and b) allowing for a coarse-grained but highly efficient discovery. Due to its flexibility, the model proposed is expected to cover a wide range of use cases.

1 Introduction

Service-Oriented Architectures (SOAs) are receiving increasing attention, as they promise an important gain in flexibility and scalability of IT systems and a reduction of the integration burden. However, the adoption of the SOA paradigm is not only expected to bring benefits but also to pose new research challenges. One of these challenges is the automatic location of Web services that can fulfill a given functional goal, and it constitutes the focus of this paper.

We concentrate on the exploitation of the formal description of Web service functional capabilities and of customer goals and, building on previous works ([13, 14, 16, 17]), we propose a Web service discovery model with some features that we believe can make it usable in a wide variety of scenarios, namely: a) it allows for different trade-offs between accuracy of results and response times, based on the application of different filters, b) it supports users in the description of their goals and Web services, c) it can provide meaningful results in short times if only the simplest filter is applied, and d) it can yield results of considerable accuracy if all available filters are successively applied. Additionally, a prototype of a registry and a discovery component implementing the model designed has been built as a proof-of-concept of our approach. This prototype is presented in the paper as well as some preliminary performance evaluation and complexity results based on the logical languages and reasoning tasks employed.

The paper is structured as follows: Section 2 introduces alternative views of Web service capabilities used in our model. The publication of Web services by providers and the description of goals by customers are presented in Sections 3 and 4, respectively. In Section 5, the application of registry-side filters for locating Web services that can resolve the goal at hand is described, and customer-side filters are presented in Section 6. Relevant related work is discussed in Section 7. Finally, our conclusions and future work are summarized in Section 8.

2 Web Service Capabilities

Web services are computational entities accessible using particular standards and protocols, and usable to request the provision of some services, being a service understood as a provision of value in some domain [13, 22]. For example, an airline can publish a Web service for the booking of its flights; this Web service can be used to request different particular bookings, each booking being a service.

The provision of a service results on the provision of some information to the customer (*service outputs*) e.g. the provision of weather information and/or on some changes in the real world (*service effects*) e.g. the actual booking of a seat on a flight; the functional value of the Web service is given by the outputs and effects associated to the services that can be requested through it. We call this functional value the functional *capability* of the Web service [23].

In our model, a provider can describe the functional capability of its Web service in different ways, namely: a) only syntactically using WSDL [5], b) assigning the Web service to one or more categories, and/or c) semantically, with different level of detail (only inputs, only results, or also the relation between both). The publisher will be free to choose among these types of description and to combine them.

For describing Web services, regardless of the type of description used, we use the WSMO framework [23] and the WSML [9] family of languages. However, our model could work with other frameworks. In this section, we present the alternative descriptions allowed in our model and how they are incorporated into the WSMO framework. We do not explain how WSDL Web services are described; for details on how WSDL is used to syntactically describe the function a Web service offers we refer to [5].

2.1 Semantic description with input-results relation

The service provided by a Web service and, thus, the results (outputs and effects) of the Web service execution depend on the information given by the customer to the Web service as *input values*, and on the state of the world that holds when a request is issued to the Web service. The booking provided by the airline Web service mentioned above will depend e.g. on the flight data given to the Web service and on seat availability, which is part of the state of the world. If we abstract from the possibly complex interaction required by Web services for providing their functionality, a Web service can be seen as a function that maps input values and the state of the world to some outputs and effects.

The first kind of description we will consider is a formal description of valid input values for the Web service and of possible results depending on the particular input values provided by the customer. We drop the dependency of Web service results on the current state of the world from the description, as the discovery process will have partial visibility of the current state of the world and, thus, a distributed, recursive evaluation of capability descriptions would be required if such dependency has to be considered by the discovery process, which can considerably degrade efficiency [16].

Figure 1 shows the description of a flight booking Web service. Required input values are encoded by a WSMO precondition: information of the flight to be booked (variable $?flight$), which must be between European cities and operated by airline *air*, and a *MasterCard* credit card (variable $?cc$), are required. The WSML language used for describing preconditions is WSML-Flight [9], as it will be explained in Section 6.

The conditions results of the Web service fulfill, dependent on the particular input values provided by the customer, are encoded by *input-dependent* postconditions, identified by the value of the *postconditionType* non-functional property. In the example, every result (variable $?result$) of the Web service will be a booking of a flight $?flight$, paid with credit card $?cc$. Variables $?flight$ and $?cc$ are declared in the capability as shared between the precondition and the postcondition and, therefore, the formalization of Web service results is dependent on the values given to these variables i.e. on the input values given by the customer, as we will see in Section 6.

The modelling style is the same one used in [16], and the description of input-dependent results is regarded as a not yet named Description Logics (DL) [20] concept definition that formalizes the conditions all results of the Web service fulfill, dependent on the input values given by the customer. The expressivity is restricted to the $\mathcal{SHOIQ}(\mathcal{D})$ DL [20], which is the description logic underlying WSML-DL with the addition of nominals, as shared variables in such concept definitions will be replaced by instances during discovery (see Section 6), and very close to the DL underlying OWL ($\mathcal{SHOIN}(\mathcal{D})$) [2]. In the following, we will denote WSML-DL *plus* nominals by WSML-DL+.

Notice that all results of the Web service, both outputs and effects, are encoded in the postcondition of the Web service capability. Outputs and effects are distinguished ontologically, as e.g. the $Booking$ concept is a subconcept of $Effect$ in the domain ontologies used, and there is a concept $InfoProvision$ for identifying the provision of information i.e. outputs. Ontologies are not shown due to space constraints.

2.2 Formal description without inputs-results relation

The second type of description we will consider is the formal description of valid input values and of possible results of the Web service, but without their relation. Remarkably, this is close to the kind of descriptions used by pure DL-based approaches to Web service discovery such as [4, 17, 21].

The description of valid input values is the same one introduced above, encoded in the precondition of the Web service of Figure 1, while the possible results of the Web service, independently of the particular input values given, are encoded by the *input-independent* postcondition of Figure 1, identified by the value of the *postconditionType*

non-functional property. Notice that there is no shared variable between the precondition and the input independent postcondition.

The description of the input-independent postcondition, in WSML-DL+, is regarded as a not yet named DL concept definition that formalizes the conditions results of the Web service fulfill, *independently of input values*. The input-dependent and input-independent formalizations of results are related in the way described in [16].

2.3 Categorization

Web services can be assigned to functional categories i.e. to categories that capture a given functional value. For example, a Web service for booking flights can be assigned to a transport means booking category, and this category reflects to some extent the functionality of the Web service. The last type of description we will consider is the assignment of the Web service to (one or more) functional categories.

In Figure 1, the category the Web service is assigned to is *TransportMeansBooking@http://www.tifbrewery.com/seta/Taxonomy1* i.e. a transport means booking category defined at taxonomy *http://www.tifbrewery.com/seta/Taxonomy1*, and the assignment is encoded by the *category* non-functional property of the capability.

Taxonomies of functional categories are intuitive for average users if properly defined i.e. with an appropriate number of categories, properly arranged, with a clear meaning and a clear textual explanation, etc. Furthermore, we associate a formal meaning to categories in taxonomies; when a taxonomy of categories is defined, we associate to each category a formal and general description of the results expected from Web services assigned to that category, independently of possible input values. For example, the following WSML description is associated to a transport means booking category:

```
?result memberOf Booking[ofitem hasValue ?item] and ?item memberOf TransportMeans.
```

The description above formalizes general results, and particular Web services assigned to this category can more precisely formalize the results they offer e.g. booking of flights, possibly including how these results depend on input values. The language used for describing these general results is also WSML-DL+, and descriptions like the one above are regarded as a not yet named DL concept definition formalizing the conditions general results of Web services in a category fulfill.

3 Web Service Publication

Service providers can make their services accessible via Web service interfaces. In order to make a Web service usable by other parties, a provider will *publish* the Web service description at some network location reachable by target users. It is a common practice to publish syntactic WSDL descriptions of Web services at UDDI [3] repositories, which act as a common entry point for the location of Web services and provide keyword-based search facilities as well as search based on categories in taxonomies such as UNSPC. However, the publication of WSDL descriptions at UDDI repositories has two major limitations: a) the assignment of Web services to categories is completely manual, and b) the use of syntactic descriptions does not allow for advanced search based on formal semantics [24].

```
webService _"http://www.tifbrewery.com/seta/FlightBooking"
  ...
  capability FlightBookingCapability
    nonFunctionalProperties
      afi#category hasValue "TransportMeansBooking@http://www.tifbrewery.com/seta/Taxonomy1"
    endNonFunctionalProperties

    sharedVariables {?flight, ?cc}

    precondition
      definedBy
         ?flight memberOf Flight[cityOrigin hasValue ?co, cityDestination hasValue ?cd,
          operatedBy hasValue air] and ?co memberOf City[inContinent hasValue europe] and
          ?cd memberOf City[inContinent hasValue europe] and ?cc memberOf MasterCard.

    postcondition
      nonFunctionalProperties
        afi#postconditionType hasValue "inputDependentPostcondition"
         afi#intention hasValue "all"
      endNonFunctionalProperties
      definedBy
        ?result memberOf Booking[ofItem hasValue ?flight, withPaymentMethod hasValue ?cc].

    postcondition
      nonFunctionalProperties
        afi#postconditionType hasValue "inputIndependentPostcondition"
         afi#intention hasValue "all"
      endNonFunctionalProperties
      definedBy
        ?result memberOf Booking[ofitem hasValue ?item, withPaymentMethod hasValue ?pm] and
               ?item memberOf Flight[cityOrigin hasValue ?co, cityDestination hasValue ?cd,
               operatedBy hasValue air] and ?co memberOf City[inContinent hasValue europe] and ?cd
               memberOf City[inContinent hasValue europe] and ?pm memberOf MasterCard.
```

Fig. 1. Example Web service

In this setting, we have implemented a registry (Figure 2) using a UDDI repository (jUDDI[1]) and a DL reasoner (RacerPro [1]), enabling the retrieval of Web services based on the semantic description of the results they offer and supporting the publisher in describing Web services. A taxonomy manager, with a Web service interface, allows for the management of category taxonomies, and a publication manager is in charge of the publication of Web services at the registry, making use of jUDDI, RacerPro, and a categorizer. Both are introduced in this section. The query manager allows for the location of Web services in the registry (see Section 5). We note that the UDDI API implemented by jUDDI can also be directly used for the syntactic search of Web services.

The registry is based on [24], but we allow for arbitrary combinations of concepts in domain ontologies to describe the results of Web services, and we incorporate the usage of taxonomies of categories.

3.1 Classification in categories

When a Web service is published at the registry, it is first assigned to one or more functional categories as it will be explained below. However, before any Web service

[1] http://www.juddi.org/

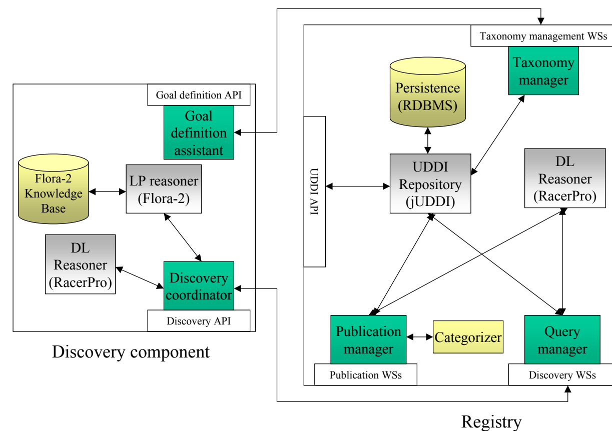

Fig. 2. Registry and discovery component architecture

can be assigned to a category, such category must be included in the registry. This is done through the taxonomy manager, which allows for the definition of taxonomies of categories and publishes them at the UDDI repository used by our registry. The definition of categories includes the formal description of the general results offered by Web services in that category (see Section 2.3).

Categorization of WSDL Web services. If the publisher only has available a WSDL description of the Web service, he can only *manually* browse category taxonomies known by the taxonomy manager and assign his Web service to (one or more) categories, as done in current UDDI repositories. If no category is selected and no further information is given by the publisher, we will have no semantic description of the Web service and it will be only published at jUDDI and searchable through the UDDI API.

If the Web service is assigned to one or more categories, the formal description of the general results expected from Web services in these categories will be retrieved and shown to the publisher so that it serves as a starting point for formally describing his Web service. The publisher can refine this description and possibly add the formal description of input values and the relation between Web service results and these values. For example, if the category chosen is *TransportMeansBooking*, the formal description retrieved will be the one given in Section 2.3, which the publisher can refine to describe e.g. that the Web service only offers the booking of flights, as done in the input independent description of results of Figure 1. In the simplest case, the customer will not refine the results of the categories used, and the formalized results of the Web service will be the intersection of the results of these categories. A WSMO description is generated to include the formal details available for publication.

Categorization of semantic Web services. If the publisher has already available not only a WSDL description but also a semantic description (of results and optionally of input values) of his Web service, our classification framework can assist him in the

assignment of the Web service to categories. The publication manager uses the categorizer in Figure 2 to heuristically, and based on the semantic description of inputs and results, propose the categories from existing taxonomies that best fit the Web service. This is done by measuring the similarity between the semantic description of the Web service inputs and results, and the inputs and results of Web services published at the registry and already assigned to a given category (see [6, 7] for details). In our current implementation, only the type descriptions (i.e. named classes from domain ontologies involved in inputs and results descriptions) are used.

After this heuristic classification, the publisher is given an ordered list of the categories his Web service would most likely belong to, thereby assisting him in its categorization. Still, the publisher will make the final choice of categories.

We have measured classification times using a repository of 164 classified semantic Web services[2]. This repository has been automatically translated from OWL-S (its current format) to WSMO. Since the classification heuristic needs a repository of classified Web services (the training set) in order to provide meaningful results, it is not possible to use randomly generated services for testing due to the cost of manually classifying them. Therefore, our tests are limited to a set of 164 Web services, obtaining a response time of around 650 milliseconds for classifying a new Web service when 156 of them (the training set) are already classified[3].

Consistency checking. Once the publisher has assigned his Web service to categories and sent its final WSMO description to the publication manager, the description of the input-independent results of the Web service and the description of the results associated to the selected categories are checked for consistency [20] so that the Web service is not assigned to categories whose general results are not consistent with the results the Web service declares. For example, we will check that the input-independent postcondition of the Web service in Figure 1 is consistent with the formalization of results of the transport means booking category in Section 2.3. If only a WSDL description was initially given, this check will only be necessary if the publisher refines the formalization of results associated to the selected categories.

Both descriptions, in WSML-DL+, are checked for consistency using RacerPro. However, RacerPro only offers sound and complete reasoning for $\mathcal{SHIQ}(\mathcal{D})$ and, therefore, nominals [20] are translated into pair-wise disjoint concepts before they are sent to the reasoner. Some incorrect inferences can be drawn from the resulting translation [11], but in most cases the subsumption and satisfiability relations computed will be correct. Recently, a sound and complete reasoning procedure for $\mathcal{SHOIQ}(\mathcal{D})$ has been implemented in the Pellet[4] reasoner; testing the use of Pellet instead of RacerPro will be part of our future work.

We have generated random variations of an initial set of Web services in order to measure the time required for consistency checking. This task is time consuming, as checking concept satisfiability is NExpTime-complete for the $\mathcal{SHOIQ}(\mathcal{D})$ DL [19]. In our tests, we have measured times of around 20 seconds for checking satisfiability

[2] From http://www.few.vu.nl/~andreas/projects/annotator/owl-ds.html

[3] For all tests in the paper, an Intel Pentium 4 2.8 GHz, 1GB RAM computer has been used.

[4] http://www.mindswap.org/2003/pellet/

of concepts referring to an ontology of around 800 concepts. However, these times are acceptable as publication is not time-critical[5].

3.2 Results-based classification

After checking consistency of category results and Web service results, the publication manager will store a business service [3] at jUDDI, which will have a unique key $key_{\mathcal{W}}$ assigned automatically by the repository. This business service includes: a) the complete WSMO description of the Web service, b) the functional categories the Web service has been assigned to, and c) the WSDL description of the Web service, if available, so that the Web service can be searched syntactically through the UDDI API.

Afterwards, the Web service will be classified in a subsumption hierarchy attending to the formal, input-independent description of its results. For this purpose, a $\mathcal{SHOIQ}(\mathcal{D})$ concept is defined, being the name of the concept the unique key $Key_{\mathcal{W}}$ given by the UDDI repository, and the definition of the concept the one given by the input-independent postcondition of the Web service. For example, the following concept is defined for the Web service from Figure 1:

?result **memberOf** Key **equivalent** ?result **memberOf** Booking[ofitem **hasValue** ?item,
withPaymentMethod **hasValue** ?pm] and ?item **memberOf** Flight[cityOrigin **hasValue** ?co,
cityDestination **hasValue** ?cd, operatedBy **hasValue** air] and ?co **memberOf** City[inContinent **hasValue**
europe] and ?cd **memberOf** City[inContinent **hasValue** europe] and ?pm **memberOf** MasterCard.

This concept is sent to the T-Box [20] of the DL reasoner (after translating nominals), and the T-Box is classified i.e. the subsumption relation between concepts in the T-Box is computed. In this way, published Web services will be arranged in a subsumption hierarchy in terms of the results they can provide, which we will be able to query efficiently. As checking subsumption can be reduced to the reasoning task of checking satisfiability, T-Box classification is also NExpTime-complete. We have measured times of around 160 seconds for classifying the T-Box when 2000 Web services are already published, and using an ontology of around 800 concepts [16]. However, classification is done off-line and, therefore, it is not a time-critical task as long as publication times are kept within certain limits.

Summarizing, when a Web service is published it will be assigned to categories, stored at the jUDDI repository, and classified at the T-Box of the DL reasoner according to the formalization of the results it can provide. Remarkably, the publisher is assisted in assigning Web services to categories and in formalizing their results based on the formal meaning associated to categories in taxonomies. In addition, Web services described with different level of detail can be published, including purely syntactic WSDL descriptions. However, the kind of semantic description given at publication time will condition how Web services can be later evaluated by the discovery process.

4 Customer Goals

Customers will be interested in using Web services because of the functional value they offer i.e. because of the outputs and effects they can provide. Therefore, customer goals

[5] We note, however, that beyond a certain number of Web services published, the times required for consistency checking can start to be too high and indeed problematic

have to describe the functional value expected from Web services. In this section, we discuss how goals are described in our model.

4.1 Description of goals

The functional value expected by a customer can be described: a) by giving functional categories in available taxonomies, or b) by formalizing the results (outputs and effects) expected. In the first case, the customer can browse category taxonomies and select the categories sought Web services must belong to. The use of categories for locating Web services is intuitive for the end-user, as he does not have to deal with formal descriptions, and, as we will see in the next section, response times are low.

However, the results obtained using categories for discovery are coarse-grained. For this reason, we also try to make the use of formal descriptions easier for customers so that they can specify their expected results in more detail if required. In particular, we use the formal descriptions associated to categories as follows: a customer can browse category taxonomies and select categories he is interested in; when these categories are selected, we retrieve the formal description of results defined for these categories so that they can be refined by the customer for accurately describing his goal. In this way, we provide the user with basic support for formalizing the results he requires.

Figure 3 shows a goal description, where a category is encoded as a non-functional property of the goal, and the results expected (the booking of a flight $myFlight$, paid with credit card $myCC$) are encoded in the postcondition. The language used for formalizing expected results is WSML-DL+.

The customer must also associate a universal or existential intention to the formalization of expected results (encoded as a non-functional property *intention*), meaning that he wants to obtain all or only some of the results from the set formalized. This intention will influence what Web services are considered a match, as explained in [13].

We have implemented a discovery tool that supports the user in describing goals by exploiting taxonomies of categories. In particular, it enables browsing taxonomies and retrieving the formal descriptions associated to categories so that they can be used as a starting point for formalizing expected results. The tool uses the discovery component of Figure 2, which is installed at the customer side. In particular, it uses its goal definition assistant, which communicates with the taxonomy manager in the registry to retrieve information about taxonomies of categories.

Level of accuracy. Once either a functional category or a formalization of expected results is given (including its intention), the second ingredient in the description of a goal is the level of accuracy expected from discovery results. In particular, the customer can choose to apply different filters to obtain Web services that can potentially fulfill his goal. We split these filters into registry-side filters, which do not take inputs into account, and customer-side filters, which are applied over the Web services obtained from the registry after applying registry-side filters and which focus on availability of input values and on how these values condition Web service results. Only customer-side filters consider input values because, as we will explain below, potential input information is kept locally by the customer.

The customer can choose to apply one of these two registry-side filters: 1) category-based, or 2) results-based. Two customer-side filters can be successively applied: a) input-availability, and b) input-results relation. Registry-side and customer-side filters will be presented in Sections 5 and 6, respectively. For formal details on the results-based, input-availability and input-results relation filters, we refer the reader to [16].

4.2 Input information

If customer-side filters are applied, we evaluate whether the customer has appropriate input values for relevant Web services, and possibly whether these input values would lead to the results required, is evaluated. In this setting, we assume a knowledge base KB exists, containing: a) the information the customer knows and he is willing to disclose for achieving his goal, and b) the domain ontologies $\mathcal{O}$ providing the domain vocabulary.

KB will be kept local, as it might be big in volume and it might include customer's sensitive information e.g. credit card details, which should not be disclosed to the registry or to any third-party at discovery time [16]. In our current prototype, KB is implemented as a Flora-2[6] knowledge base which will be queried as described in Section 6, and which is defined once and used for resolving all goals issued by the customer. The information in the Flora-2 knowledge base will also be stored in the DL reasoner of the discovery component (see Figure 2).

Regarding the expressivity allowed for describing information in KB, ontologies in $\mathcal{O}$ must be described using WSML-Core [9] and, thus, restricted in expressivity to Description Logic Programs (DLP) [10]. Furthermore, the information the customer knows must be described as instances of concepts in $\mathcal{O}$ and, thus, as instances of WSML-Core concepts. In this way, we can consistently refer to instances of the same domain ontologies from WSML-Flight and WSML-DL+ descriptions (see [9, 16] for details), and use the same domain model and customer knowledge in Flora-2 and RacerPro. Remarkably, and according to [26], most ontologies in popular ontology libraries fall in the DLP fragment of first-order logic.

In a nutshell, goals are defined by either categories in category taxonomies or a WSML-DL+ formalization of expected results, plus the filters that must be applied for finding relevant Web services (encoded by the *filter* non-functional property of the goal). Additionally, if customer-side filters are selected, a knowledge base KB containing the information on the customer side that can be used as input values to relevant Web services must be in place.

5 Registry-side Filters

The discovery component depicted in Figure 2 is installed at the customer side, and the customer communicates with it to resolve his goal. In particular, the customer submits his goal to the discovery coordinator of Figure 2, which first sends it to the registry query manager using the Web service interface exposed for this purpose. The query

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

```
goal _"http :// www.tifbrewery.com/seta/FlightBookingGoal"
  ...
  capability FlightBookingGoalCapability
    nonFunctionalProperties
      afi#category hasValue "TransportMeansBooking@http://www.tifbrewery.com/seta/Taxonomy1"
        afi#filter   hasValue {"ResultsBased", "InputAvailability"}
    endNonFunctionalProperties

    postcondition
      nonFunctionalProperties
        afi#intention  hasValue "some"
      endNonFunctionalProperties
      definedBy
        ?result  memberOf Booking[ofitem hasValue myFlight, withPaymentMethod hasValue myCC].
```

Fig. 3. Example Goal

manager will apply the registry-side filter selected and return relevant results to the discovery coordinator, as it will be explained in the following.

5.1 Category-based filter

If the category-based filter is selected, the query manager extracts the categories specified by the *category* non-functional property, which are the categories sought Web services must belong to (interpreted as a logical *and*).

As Web services in the registry are already categorized, the query manager will simply invoke the UDDI *find_service* operation [3] specifying the categories given by the goal, and the keys of Web services assigned to these categories will be retrieved. Afterwards, the query manager will query the UDDI repository through the *get_serviceDetail* operation, specifying the keys of relevant Web services, in order to retrieve their complete descriptions. The complete descriptions of the obtained Web services, categorized as required by the goal, are returned to the discovery coordinator.

The operations above perform simple lookups in the relational database used for persistency of the UDDI repository and, thus, response times are low. According to our tests, response times are below 300 milliseconds for 2000 Web services categorized at the registry and remain almost constant. While results will be provided in short times, and based on the use of categories intuitive for end-users, they will be coarse-grained, as the granularity of results will be limited by the granularity of categories in taxonomies.

5.2 Results-based filter

If the results-based filter is applied instead, the query manager will query the DL reasoner T-Box for concepts: a) equivalent to, b) more general than, c) more specific than, and d) with a non-empty intersection with the description of the set of results expected by the customer, given by the postcondition in the goal description after replacing nominals i.e. concepts describing sets of offered results related in some way to the results expected are retrieved. In practice, as RacerPro does not allow for querying for *all* concepts intersecting a given concept, we query for Web services *not intersecting* the set of results expected. Intersecting concepts will be obtained by taking all available concepts

representing Web service results but those not intersecting the goal and those having a subsumption (equivalent, more general, or more specific) relation with the goal. Once these concepts are retrieved, we will use the matching notions in [13] to determine which of them are a perfect, possible perfect, partial, or possible partial match for the goal, taking into account intentions.

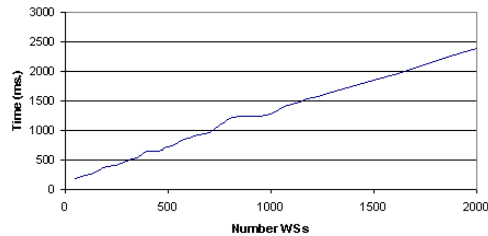

Fig. 4. Times for the application of the results-based filter

The keys of matching Web services are obtained from the name of the concepts retrieved, and jUDDI is queried for the complete description of Web services with these keys. The query manager returns to the discovery coordinator four lists with complete Web service descriptions, one per type of match. In Figure 4, the time required for retrieving Web services from the registry applying the results-based filter is shown as a function of the number of Web services published with an input-independent description of their results. A remark is in place: querying RacerPro for equivalent, more general or more specific concepts yields response times below 20 milliseconds for 2000 Web services published (see [16]), while querying for non-intersecting concepts (in order to afterwards obtain intersecting concepts) is the most time-consuming operation and increases with the number of available Web services.

6 Customer-side Filters

The application of a registry-side filter results in a number of relevant Web services being returned to the discovery coordinator. If the results-based filter is applied, four lists of Web services are returned, corresponding to the degrees of match identified in [13]. If the category-based filter is applied, a single list is returned, being all Web services considered perfect matches. If no customer-side filter is chosen, the discovery coordinator directly provides these Web services to the customer. Otherwise, customer-side filters are applied as explained below.

6.1 Input availability filter

Besides checking whether a Web service is relevant for solving a customer's goal, either in terms of the categories it belongs to or in terms of the results it can provide, the customer might want to know whether the input values required by the Web service

can be provided i.e. whether the customer has appropriate information available to be submitted to the Web service as input values.

For this purpose we will use the preconditions of Web services returned after the application of a registry-side filter as queries to the knowledge base KB formed by the customer knowledge and the domain knowledge i.e. we query for valid input values for the Web service. For example, the precondition in Figure 1 will be used as a query for values for variables $?flight$ and $?cc$. Preconditions are expressed in WSML-Flight and, thus, they have Logic Programming (LP) semantics [18]. As discussed in [16], LP semantics are appropriate in this setting as we want to determine whether valid input values can be provided to the Web service *from knowledge base* KB (close world) i.e. whether there is, at the customer side, information to be provided as input values to the Web service. Furthermore, for obtaining valid input values we are only interested in efficient query answering, not in general entailment.

The knowledge base KB, implemented as a Flora-2 knowledge base, is initialized before any goal is issued to the discovery component. Therefore, it is already compiled and loaded when a goal has to be resolved, which makes querying for valid input values using the description given by Web service preconditions efficient (less than 50 milliseconds for a knowledge base with more than 14000 randomly generated facts). In fact, query answering for WSML-Flight (Datalog with stratified negation and inequality) can be done in polynomial time [8].

Input availability is a boolean filter i.e. the input information requirements of a Web service are either fulfilled or not. However, as we allow providers to publish Web services with different level of detail in their description, there might be Web services retrieved from the registry that do not describe the input values they require. Therefore, the discovery coordinator will return to the customer: 1) Web services that passed the input availability filter, and 2) those to which the filter could not be applied. Web services will be grouped into these categories, and their complete descriptions will be returned to the customer along with their degree of match given by the application of registry-side filters.

For Web services that pass the input availability filter, we obtain what valid input values were found. In particular, the queries described by Web service preconditions yield a set of assignments of instances in KB to input variables. In particular, if the set of input variables is $I = \{i_1, \ldots, i_n\}$, different sets of assignments of instances to variables can be obtained, being each such set denoted by $I[j]$. If we consider the Web service in Figure 1, we could assign e.g. some instances $myFlight$ and $myCC$ to variables $?flight$ and $?cc$ (assignment set $I[1] = \{flight/myFlight, cc/myCC\}$), or some instances $myFlight$ and $myCC2$ (assignment set $I[2] = \{flight/myFlight, cc/myCC2\}$), where $myFlight$ is a flight between European cities operated by airline air, and both $myCC$ and $myCC2$ are MasterCard credit cards. These instances, and the complete knowledge base KB, are available both to the DL and the LP reasoners.

As a WSML reasoner is not completely available yet, we use directly Flora-2. For this purpose, we need to translate WSML-Flight descriptions to Flora-2 syntax. This is currently done manually before publishing Web services, and preconditions are encoded using Flora-2 syntax besides WSML syntax in Web service descriptions [16].

The same applies to the use of RacerPro: we currently translate WSML-DL+ descriptions to RacerPro syntax and incorporate descriptions in both syntaxes into goal and Web services.

6.2 Input-results relation filter

As discussed in Section 2, there is a relation between Web service results and the input values provided by the customer. If the input-results relation filter is selected, we will not only check whether the customer has available appropriate input values for relevant Web services retrieved from the registry, but also the relation between these input values and Web service results.

Given the assignment sets obtained from applying the input-availability filter, we can replace shared variables in the input-dependent postcondition (which are part of the set of input variables) by input values, yielding a WSML-DL+ concept definition. Notice that, as these values are instances of WSML-Core ontologies, we can consistently use them in WSML-DL+ descriptions. Each assignment set, corresponding to the usage of the Web service with different input values, is considered. The concept defining the results offered by a Web service for available input values is defined by the union of the results for each such set of input values. If we continue with the example above, the assignment sets $I[1]$ and $I[2]$ yield a concept WS (where WS is the Web service identifier) as follows:

?result **memberOf** WS **equivalent** ?result **memberOf** Booking[ofItem **hasValue** myFlight,
withPaymentMethod **hasValue** myCC] or ?result **memberOf** Booking[ofItem **hasValue** myFlight,
withPaymentMethod **hasValue** myCC2].

This concept will be published at the T-Box of the DL reasoner used by the discovery component (Figure 2) after replacing nominals. The discovery coordinator will then query for concepts: a) equivalent to, b) more general than, c) more specific than, and d) intersecting with the description of the set of results expected by the goal (with nominals replaced). The corresponding intentions will be applied as described in [13], yielding the list of Web services that are a perfect, possible perfect, partial, or possible partial match for the goal for the input values available. In a nutshell, we obtain Web services that offer, *for the input values available*, the results expected by the customer, considering how these results depend on the input values provided i.e. we evaluate the relation between input values and Web service results.

This filter can only be applied if the goal formalizes the set of results expected, as this formalization must be used in the queries to the DL reasoner for relevant Web services. Furthermore, as Web services might not describe input-dependent postconditions, the discovery coordinator returns to the customer: 1) Web services that passed the input-results relation filter, 2) those that passed the input availability filter but to which the input-results relation filter could not be applied, and 3) those to which none of the filters could be applied. Web services will be grouped into these categories, and their complete descriptions are returned to the customer along with their degree of match.

Besides querying the Flora-2 knowledge base for valid input values, the DL concepts obtained from replacing parameters with these values have to be published and the DL reasoner has to be queried for concepts having certain relation with the results required by the goal. This querying is time-consuming, as new concepts are published

during discovery and, thus, the T-Box of the DL Reasoner cannot be fully classified before-hand. While we expect most Web services to be discarded by previous filters so that not many Web services have to be evaluated in this way, the times required are still high e.g. around 60 seconds for 500 Web services evaluated. Therefore, this filter should only be applied if a detailed evaluation of Web services is needed.

7 Related Work

Most (semantic) web service discovery proposals are based on the semantic matching of Web services and goals based on DL reasoning e.g. [4, 17, 24], focusing on a single logical filter and not considering neither the relation between input values and Web service results, nor alternative filters.

Works such as [14] and [12] take into account how the results of using a Web service depend on the input values provided to it. The former makes use of Transaction Logic and Logic Programming, which presents some problems with the treatment of unspecified information in goals and Web service capability descriptions. The latter, based on DL, expects customers to specify what kind of relation they expect between results and inputs, which we believe is a less usable approach than that taken by our model. None of them explicitly considers the combined application of other filters.

The only approaches we are aware of that apply different types of filters are LARKS [25] (for the matching of agents) and OWLS-MX [15] (for the matching of OWL-S Web services). While they are close in spirit to our model, allowing for the application of different types of filters, they differ in the following major respects: a) logical filters are restricted to first-order semantics, b) whether the customer can actually provide appropriate input values is not evaluated, c) the relation between available input values and Web service results is not described and exploited, d) all matching is done at the registry side, and e) customers are not supported in describing goals and Web services.

It must be noted that the type of filters considered by OWLS-MX could be incorporated to our model as registry-side filters. We are currently adding to our prototype a filter that matches textual descriptions of goals and Web services, but only based on the simple keyword matching provided by jUDDI; nevertheless, extending it in the direction of more complex syntactic similarity measures is a feasible task.

8 Conclusions and Future Work

We have presented a model and a prototype that accommodates different levels of accuracy in the description of Web services and goals, and offers discovery results with different trade-offs between accuracy and efficiency, so that it is usable in different application scenarios. We also incorporate the use of taxonomies of categories in order to provide basic assistance to the user in the semantic description of Web services and goals, and in order to obtain coarse-grained discovery results in short times. Future work will concentrate on optimizing the discovery prototype, on the refinement of the heuristic classification of Web services in taxonomies, on the proposal of new filters and the integration of existing ones such as those used by OWLS-MX, and on studying possible extensions of the expressivity allowed for domain ontologies.

References

1. RacerPro user's guide. version 1.9. Technical report, RACER Systems GmbH & Co. KG, December 2005.
2. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Technical report, W3C Recommendation, Feb 2004.
3. T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, Maryann Hondo, Y.L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI Version 3.0, 2002.
4. B. Benatallah, M. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *VLDB*, 14:84–96, 2005.
5. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.
6. M. A. Corella and P. Castells. A heuristic approach to semantic web services classification. In *KES-2006*, 2006.
7. M. A. Corella and P. Castells. Semi-automatic semantic-based web service classification. In *semantics4ws'06 workshop at BPM 2006*, 2006.
8. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of Logic Programming. *ACM Computing Surveys (CSUR)*, 33(3):347–425, 2001.
9. de Bruijn, J. (ed.). The Web Service Modeling Language WSML. WSML d16.1v0.21, 2005.
10. B.N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *WWW'03*, 2003.
11. I. Horrocks and U. Sattler. Optimised Reasoning for $\mathcal{SHIQ}$. In *ECAI 2002*, pages 277–281, July 2002.
12. D. Hull, E. Zolin, A. Bovykin, I. Horrocks, U. Sattler, and R. Stevens. Deciding semantic matching of stateless services. In *AAAI-06*, 2006.
13. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *ESWC 2005*, Heraklion, Greece, May 2005.
14. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *SWSs Worshop at ISWC 2004*, 2004.
15. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *AAMAS 2006*, 2006.
16. R. Lara. Two-phased web service discovery. In *AI-driven Service Oriented Computing workshop at AAAI 2006*, July 2006.
17. L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *WWW'03*, Budapest, Hungary, May 2003.
18. J. W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.
19. C. Lutz. An improved nexptime-hardness result for description logic alc extended with inverse roles, nominals and countins. Technical report, Technical University Dresden, 2004.
20. D. Nardi, F. Baader, D. Calvanese, D. L. McGuinness, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge, January 2003.
21. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *ISWC 2002*, pages 333–347. Springer Verlag, 2002.
22. C. Preist. A Conceptual Architecture for Semantic Web Services. In *ISWC 2004*, 2004.
23. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, D. Fensel, and C. Bussler. Web Service Modeling Ontology. *Applied Ontology Journal*, 1(1), 2005.
24. N. Srinivasan, M. Paolucci, and K. Sycara. Adding OWL-S to UDDI, implementation and throughput. In *SWSWPC Workshop at ICWS 2004*, 2004.
25. K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *AAMAS 2002*, 2002.
26. Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, 2004.