A flexible model for the location of services

Ruben Lara[1] Miguel Corella[1]
Pablo Castells[2]

[1] *Tecnología, Información y Finanzas, Madrid, Spain*

{rlara, mcorella}@afi.es

[2] *Universidad Autónoma de Madrid*

pablo.castells@uam.es

May 15, 2007

Abstract

The advent of the SOA paradigm is expected to cause an increase in the number of available services. This increase can create a bottleneck for the location and, therefore, for the use of services that provide a particular value. In this paper, we propose a model for the effective location of services, and a prototype implementation of this model, with some features which make it applicable to heterogeneous usage scenarios, namely: 1) it is flexible in the kind of descriptions of services and goals expected, 2) it includes assistance to consumers and providers in describing the services sought and offered, respectively, and 3) it offers different trade-offs between the accuracy of results and the efficiency of the location process based on the application of different filters.

1 Introduction

SOA is attracting increasing attention as an architectural paradigm that can enable a higher reuse of IT assets, their principled and eased integration into more complex services and, therefore, a more agile adaptation and evolution of IT systems to respond to business needs.

The exposition of functionalities provided by heterogeneous and possibly distributed systems as reusable, platform-independent, interoperable, and meaningful (from a business point of view) services is the pillar of SOA; existing and new pieces of functionality are given visibility and exposed so that

they can be seamlessly accessed by other parties. However, for services to be used, they must be first located. As SOA adoption increases and more services are available, the difficulty of locating an appropriate service for solving some particular need can become a bottleneck for the effective exploitation of services unless appropriate location mechanisms are in place. The need for such mechanisms arises in different types of usage scenarios, namely: a) location of services at design-time for their composition or integration into more complex systems or processes, b) run-time location and usage of services, and c) location of services by end, human users.

For example, a telecommunications provider might define a new business process for informing their customers about special offers and promotions. This business process might involve activities realizable by using available services such as notifying customers by different means (SMS, e-mail, etc.) or registering what offers and promotions each customer was informed of. The professionals in charge of the definition and implementation of this process will want to locate, at design-time, appropriate services that can be statically incorporated into the process for performing these activities -case a) above-. Furthermore, if any of the services statically bound to the process fails at run-time, it is desirable to dynamically locate a new service which can replace the failed service -case b)-.

Finally, end, human users will want to locate services which can be used to fulfill their objectives -case c)-. For example, if a user wants to contract some of the promotions offered by the telecommunications provider, he will want to locate a service which can be used to perform the desired contracting.

In this paper, we propose a model and a prototype implementation for (semi)automating the location of services based on the value they offer. Given the variety of particular usage scenarios of different types we can find, our model has been driven by the following main considerations:

1. Different usage scenarios are expected to pose different requirements on the efficiency of the location process and on the accuracy of location results. Therefore, our model has been designed so that users can choose among different trade-offs between accuracy and efficiency based on the application of alternative filters with different properties.

2. What filters can be applied depends on the type of descriptions of services and goals available. Therefore, we must allow for alternative descriptions (and views) of goals and services so that different matchmaking mechanisms can be applied.

3. Users with different profiles might want to offer or locate services, and they will usually be able and/or willing to provide certain types of

descriptions but not others. For example, an average user might not be able to provide a formal description of its service functionality unless he receives appropriate support. For this reason, we let users choose the complexity of the descriptions they provide and, furthermore, we offer support to them in describing their goals and services.

Given these considerations, we propose an extensible set of alternative descriptions of the value of services (Section 2) and of consumers' goals (Section 3). In Section 4, different strategies for assisting users in describing their services and goals, and the process of publishing service descriptions at a registry accessible by interested parties, are described.

We also propose different matchmaking mechanisms over alternative descriptions of goals and services. Such matchmaking mechanisms will have different properties in terms of response times and accuracy and, following the spirit of LARKS [31], will be regarded as filters which determine the set of services usable to fulfill a goal, and whose application is decided by users depending on their particular needs. These filters will be split into two groups: filters that can be applied at the registry (Section 5), and filters whose application must be done once service descriptions are retrieved from the registry, i.e., at the consumer's side (Section 6). Finally, we will discuss relevant related work in Section 7 and present our conclusions in Section 8.

2 Types of descriptions of services

In this section, we explain how the concepts of service capability and service functionality are understood in our model, and we introduce alternative descriptions of these elements.

2.1 Services

Services in an SOA are computational entities usable to access a certain *capability* [25], being a capability understood as the ability of performing some action with a perceived value, in the sense that it can constitute a (perhaps partial) solution to some problem. For example, a party can have the capability of booking seats on flights operated by a given set of airlines, and this capability can be made accessible via a WSDL [5] service.

A capability has associated certain *effects*, that is, results of using a capability, and the purpose of using a capability is to realize all or part of such effects [25]. These effects can be either *information effects*, i.e., some information is made visible to the party using the capability, or *real world effects*,

i.e., there is a change in a state shared by (at least) the party providing the capability and the party using the capability. For example, the capability above might have associated the effect of providing a confirmation of the booking to the consumer (information effect), and the effect of booking a seat on a given flight (real world effect).

A service thus offers access to some capability, called the *service capability*, and the service itself is a means to achieve the effects of such capability. In particular, the execution of a service results on the realization of some of the effects of the service capability. However, for such execution to happen, a service usually requires some values to be assigned to a set of input variables $i_1, \ldots, i_n$ defined by the *service interface*. Such information must fulfill certain conditions, called *information preconditions* e.g. the value assigned to i_1 must correspond to a European city. Furthermore, certain conditions beyond the information requirements of the service must hold in the shared state for a service execution to happen, called *real world preconditions*. For example, the service might require the consumer to be a registered user.

The particular information and real world effects achieved by executing a service will actually depend on the input values provided by the consumer and on the real world conditions that hold when a request is issued to the service. For example, the booking obtained by executing a service that provides access to the capability above will depend on the flight data given by the consumer of the service, and on seat availability on such flight, which is part of the real world conditions that currently hold. If we consider deterministic services, a service can be seen as a function that maps certain real world conditions and certain assignments of values to input variables, to some information and some real world effects of the service capability. This function is what we call the *service functionality*.

2.2 Alternative descriptions

Existing proposals such as [3, 22] rely on a single way of describing the value of services. However, the variety of usage scenarios in which locating appropriate services for fulfilling a goal might be required makes it advisable to allow for alternative descriptions of service capabilities and functionalities. In this way: a) providers can choose, depending on their skills and resources, what types of descriptions of their services they will provide, and b) alternative matchmaking mechanisms with different properties can be applied.

We consider the use of both formal and non-formal descriptions of services. Descriptions of services will be provided using the WSMO framework [10], and formal descriptions will be given using the WSML family of languages [11], which has the following variants:

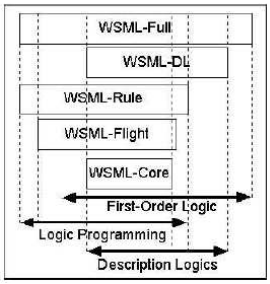

Figure 1: WSML Layering [11].

WSML-Core is the basic language of the family. It is defined by the intersection of Description Logic and Horn Logic (without function symbols and without equality), based on Description Logic Programs [14], and can thus function as the basic interoperability layer between both paradigms. In particular, ground entailment in WSML-Core coincides under first-order [13] and Logic Programming (LP) [23] semantics.

WSML-DL extends WSML-Core to the $\mathcal{SHIQ}$ Description Logic. The motivation for restricting this variant to $\mathcal{SHIQ}$ was its implementability. However, there now exist efficient implementations which can deal with nominals and, furthermore, it is desirable to be compatible with OWL DL [2]. Therefore, we will consider a new variant of WSML, denoted WSML-DL+, which is semantically equivalent to $\mathcal{SHOIN}$.

WSML-Flight extends WSML-Core in the direction of Logic Programming, and it allows for writing down any Datalog rule, extended with inequality and (locally) stratified negation.

WSML-Rule extends WSML-Flight to a fully-fledged Logic Programming language, including function symbols and without restrictions on the use of variables in logical expressions.

WSML-Full unifies all WSML variants under a common First-Order umbrella with non-monotonic extensions which allow to capture nonmonotonic negation of WSML-Rule.

WSML has been chosen as it provides an interesting set of languages with different expressivity and semantics but with two alternative layerings (see Figure 1), both complete syntactically and semantically (wrt. entailment of ground facts). The first layering is WSML-Core > WSML-Flight > WSML-Rule > WSML-Full. The second layering is WSML-Core > WSML-DL > WSML-Full, which is a complete semantic layering, also with respect to entailment of non-ground formulae.

The choice of WSMO is mainly motivated by its strong connection to WSML, which makes the integration of our WSML descriptions into the

framework easy. However, the use of other frameworks such as OWL-S [7] or SWSF [1] is possible, as long as the (extensible) set of alternative descriptions of services considered in our platform can be incorporated into service descriptions.

2.2.1 Syntactic descriptions

The use of syntactic descriptions, i.e., of descriptions without formal semantics for the location of services offers limited precision, but these descriptions are in general easier to provide by users than formal descriptions and, furthermore, low response times can be obtained when exploiting them for the location of services. For this reason, we will consider in our platform the use of the syntactic descriptions described below.

WSDL description. Services are commonly described using WSDL, which enables their execution and provides basic information about the service functionality (operations, types of inputs and outputs, and optional textual documentation), often used for locating services and deciding on their use. In order to keep compatibility with current practices and enable a smooth transition to the use of new service location mechanisms, we consider the use of WSDL descriptions.

Listing 1 shows an example WSMO description of a service which offers the booking of flights operated by airline *air* with a credit card of type *DummyCard*. The *webService* WSMO element [29] is used for representing the service, and an URL referencing the location of the WSDL description of the service is given as the value of the *wsdlDescription* non-functional property of the *capability* element[1].

Textual description. The value of a service can be described using natural language. Although WSDL descriptions can incorporate textual *documentation* elements, we will consider a textual description separate from WSDL documentation as WSDL documentation might not be available and often mixes technical documentation with the description of the value of the service. This separate textual description is encoded by the non-functional property *description* of the WSMO capability element (see Listing 1).

Categorization. The capability of a service can be described by specifying one or more categories the service belongs to. We assume the existence of taxonomies of categories *reflecting the capability of services*. For example, a taxonomy *http://www.afi.es/Taxonomy1* can be defined, including categories such as *FlightBooking*, *LowCostFlightBooking*, *CarRental* or *TrainBooking*;

[1]This encoding is used as no well-defined mechanism for incorporating WSDL descriptions into WSMO is currently available.

these categories reflect capabilities such as booking of seats on flights, booking of seats on low-cost flights, rental of cars, etc.

In Listing 1, the *category* non-functional property of the capability says that the service belongs to the category *FlightBooking* defined by a taxonomy *http://www.afi.es/Taxonomy1*, i.e., that the service has the capability of booking flights. More than one category can be specified, meaning that the service belongs to all the categories given. For example, if a service is categorized under *FlightBooking* and *CarRental*, it means that the service can be used to book seats on flights *and* to rent cars.

We will assume services are categorized using the most specific categories which fit the service capability. For example, if a service offers the booking of seats on flights operated by a low-cost airline, it will be categorized under *LowCostFlightBooking* and not under *FlightBooking*.

2.2.2 Formal descriptions

Different views of the value of a service can be formalized, yielding alternative types of formal descriptions of the service which might also have different semantics. In our setting, we will use first-order or LP semantics depending on the aspect of the service we want to capture and on how we want to exploit each type of description during the location process.

All formal descriptions will make use of a set of domain ontologies that provide the necessary domain vocabulary. This set of domain ontologies will include the definition of *actions* e.g. *Booking*, *Rental*, or *Registration*, subconcepts of *RealWorldEffect*, and of an action *InfoProvision* denoting the provision of information to consumers, i.e., information effects. These actions will be used for the description of the effects of services.

For convenience, and in order to avoid as much as possible the multiple definition of the same domain vocabulary under different semantics, we will assume the existence of a common set $\mathcal{O}$ of domain ontologies described in WSML-Core. By restricting the language to WSML-Core, the domain elements introduced in $\mathcal{O}$ will be consistently usable in descriptions in any WSML variant (and its associated semantics).

However, the expressivity allowed by WSML-Core will not be enough in certain situations; certain types of descriptions, which make use of WSML-DL+ and which will be exploited for subsumption reasoning, will require the formalization of aspects of the domain e.g. disjointness of concepts which are not expressible in WSML-Core. For this reason, we will allow for the extension of domain ontologies in $\mathcal{O}$ in the direction of Description Logics, yielding a set $\mathcal{O}^{DL}$ of domain ontologies which is the result of extending ontologies in $\mathcal{O}$ with definitions expressible in WSML-DL+ but not in WSML-Core e.g.

axioms establishing the disjointness of concepts defined in $\mathcal{O}$.

```
webService _"http://www.afi.es/AirBooking"
  capability AirBooking
    nonFunctionalProperties
      afi#wsdlDescription hasValue "http://www.afi.es/AirBooking.wsdl"
      afi#category hasValue "http://www.afi.es/Taxonomy1#FlightBooking"
      dc: description hasValue "Booking of flights operated by air with DummyCard"
    endNonFunctionalProperties

    sharedVariables {?f, ?p, ?cc}

    postcondition
      nonFunctionalProperties
        afi#postconditionType hasValue "setBasedCapability"
        afi#intention hasValue "all"
      endNonFunctionalProperties
      definedBy
       ?x memberOf Eff_AirBooking equivalent ?x memberOf Booking and
         exists ?i(?x[ofItem hasValue ?i]) and
         forall ?i(
           ?x[ofItem hasValue ?i] implies ?i memberOf Flight and ?i[operatedBy hasValue air]) and
         exists ?cc(?x[withPaymentMethod hasValue ?cc]) and
         forall ?cc(
           ?x[withPaymentMethod hasValue ?cc] implies ?cc memberOf DummyCard).

    precondition
      definedBy
       ?f memberOf Flight[operatedBy hasValue air] and ?p memberOf person and ?cc memberOf
            DummyCard.

    postcondition
      nonFunctionalProperties
        afi#postconditionType hasValue "inputDependentEffects"
        afi#intention hasValue "all"
      endNonFunctionalProperties
      definedBy
       ?x memberOf InputEff_AirBooking equivalent ?x memberOf Booking[ofItem hasValue ?f,
            forPerson hasValue ?p, withPaymentMethod hasValue ?cc].
```

Listing 1: Excerpt of the description of an example service

Set-based modelling of capabilities. The first type of formal description we consider is the description of the service capability, i.e., of the set of effects that can be achieved by using the service independently of initial conditions. This set, following our work in [18], will be formalized using first-order semantics in order to properly deal, at matchmaking time, with missing information in the description of the set.

In particular, for a service $\mathcal{S}$, we will model the set of effects achievable by a DL concept $Eff_{\mathcal{S}}$. This concept is defined by specifying what actions, from the ontologies of actions mentioned above, the service can perform and under what restrictions, as this will determine the possible effects of service usage. For example, the possible effects of using the service in Listing 1, which we will denote $AirBooking$, can be defined by a concept (in terms of

domain ontologies in $\mathcal{O}^{DL}$):

$Eff_{AirBooking} \equiv Booking \sqcap \exists ofItem \sqcap \forall ofItem.(Flight \sqcap operatedBy.\{air\}) \sqcap$
$\exists withPaymentMethod \sqcap \forall withPaymentMethod.DummyCard$

Intuitively, the formalization above says that possible effects of using the service will be bookings of flights operated by airline *air* and paid with a credit card of type *DummyCard*. This formalization is restricted in expressivity to WSML-DL+, as it corresponds to a decidable fragment of first-order logic ($\mathcal{SHOIN}$) for which reasoners offer efficient subsumption reasoning [26].

For convenience, only one concept is defined for both real world and information effects; whether the effects achievable are of one type or the other will be distinguished by the definitions given by ontologies of actions. For example, we know that the set of effects defined by $Eff_{AirBooking}$ are real world effects because in an ontology of actions the concept $Booking$ is defined as a subconcept of $RealWorldEffect$; if a service $AirSearch$ provides information about flights operated by *air* (information effects), the action used in the definition of the corresponding concept will be $InfoProvision$.

The concept that formalizes the set of effects achievable by using the service will be encoded by a postcondition of the WSMO capability, with value *setBasedCapability* for the *postconditionType* non-functional property (see Listing 1, where the concept definition is written in WSML human-readable syntax [11]).

Besides formally describing the set of possible effects of service execution, we also describe the *intention* of such description [18]: the service offers either *all* the effects in the set defined, encoded by the value *all* of the *intention* non-functional property of the postcondition, or only *some* of these effects, encoded by the value *some*. For the example above, this means that the service can provide the booking of any flight operated by *air* with any credit card of type *DummyCard* (*all*), or only some of these bookings (*some*).

Description of information preconditions. We can describe what information must be provided to a service for its execution, and what conditions such information must fulfill. In particular, and given a service $\mathcal{S}$ whose interface defines input variables $i_1, \ldots, i_n$, the values required to be bound to these variables can be described by an LP query. This query does not only describes what values are valid bindings for input variables, but it can also be used to obtain possible valid bindings for the service from a given knowledge base (see Section 6).

If we consider the service in Listing 1, with input variables f, p, cc, information preconditions can be described by the following LP query (in terms of the set $\mathcal{O}$ of ontologies):

$? - flight(F), operatedBy(F, air), person(P), dummyCard(CC)$

The definition above states that values for input variables f, p, cc are required, and that such values must be a flight operated by airline *air*, a person, and a *dummyCard* credit card, respectively. In Listing 1, this query is encoded, in WSML human-readable syntax, as a precondition of the WSMO capability element. This definition is restricted in expressivity to a WSML-Flight query [11], i.e., to a query in Datalog with inequality and (locally) stratified negation. Therefore, this type of description can only consistently refer to the WSML-Core fragment of ontologies ($\mathcal{O}$), but not to its extension in the direction of Description Logics ($\mathcal{O}^{DL}$).

The query above can be used to, given a knowledge base KB whose expressivity will be restricted to WSML-Core as we will see in Section 3.3, i.e., to the intersection of Description Logic and Horn Logic, extract valid bindings $\beta = \{f = v_f, p = v_p, cc = v_{cc}\}$ of values v_f, v_p, v_{cc} in KB to input variables, or to evaluate whether a certain input binding $\beta' = \{f = v'_f, p = v'_p, cc = v'_{cc}\}$ is a valid input binding for the service.

Description of input-dependent effects. In most cases, there is a dependency between the values assigned to the input variables of a service and the effects such service offers. We will want to describe what set of effects can be achieved by using a service with particular input values and, for this purpose, we will introduce input variables in the formalization of the set of possible effects achievable by using the service.

For example, service $AirBooking$ will book the flight and for the person given as input values to the service, and the booking will be paid with the credit card provided. We will define a concept $InputEff_{AirBooking}$ which refers to input variables f, p, cc as follows:

$InputEff_{AirBooking} \equiv Booking \sqcap$
$ofItem.\{f\} \sqcap forPerson.\{p\} \sqcap withPaymentMethod.\{cc\}$

If a particular binding $\beta = \{f = v_f, p = v_p, cc = v_{cc}\}$ is given, assigning instances (of concepts defined by $\mathcal{O}$) v_f, v_p, v_{cc} to input variables f, p, cc, we will obtain the following concept definition:

$InputEff_{AirBooking,\beta} \equiv Booking \sqcap$
$ofItem.\{v_f\} \sqcap forPerson.\{v_p\}) \sqcap withPaymentMethod.\{v_{cc}\}$

This concept formalizes what set of effects can be achieved by using the service with input values v_f, v_p, v_{cc}. In general, the definition resulting from substituting input variables by instances of WSML-Core concepts in $\mathcal{O}$ must be within the expressivity of WSML-DL+.

In Listing 1, the concept above is given, using WSML human-readable syntax, as a WSMO postcondition of type *inputDependentEffects*, and input variables f, p, and cc are declared as *shared* variables, i.e., variables f, p and cc in the description of information preconditions and of input-dependent

effects are considered occurrences of the same variables. Furthermore, the intention of the input-dependent postcondition (*some* or *all*) is encoded by the *intention* non-functional property of the postcondition.

Notice that, together, the description of information preconditions and of input-dependent effects gives a simplified view of the service functionality, without considering real world preconditions. The reasons why real world preconditions and how they influence service effects are currently not considered will be discussed in Section 6.

2.3 Relations between descriptions

Among the alternative descriptions of services introduced above there some relations of interest.

Relation between formal descriptions. The following relation must hold for any service $\mathcal{S}$ with input variables $i_1, \ldots, i_n$:

$InputEff_{\mathcal{S},\beta_i} \sqsubseteq Eff_{\mathcal{S}}$

being β_i any valid (as defined by information preconditions) input binding for $\mathcal{S}$. This means that the set of effects achievable for valid input values will be a (not necessarily strict) subset of the set of effects of the service capability.

Descriptions associated to categories. Categories can be seen as pre-defined, coarse-grained views of service capabilities: they also describe the capability of a service but with an accuracy restricted by the categories defined at available taxonomies. In fact, categories can and will be associated in our model a formal description of the capability they represent. For example, the *FlightBooking* category used in Listing 1 is associated the following definition of its possible effects (in DL syntax):

$Eff_{FlightBooking} \equiv Booking \sqcap \exists ofItem \sqcap \forall ofItem.Flight$

The formalization above corresponds to the set-based modelling of the effects associated to the category, and it will be limited in expressivity to WSML-DL+. The categorization of a service can be thus seen as a coarse-grained description, with an associated formal meaning, of its capability. Furthermore, categories can also be associated a textual description and a *prototypical* formal description of information preconditions and input-dependent effects. For example, it is usual for the booking of flights that the details of the flight to be booked must be given, as well as the details of the person for which a seat must be booked and a payment method. The effect will generally be the booking of a seat on the flight given, for the person given, and paid with the payment method specified.

3 Types of descriptions of goals

Consumers are interested in locating and using certain services because they can provide effects that solve a given consumer's need. For example, a consumer might need to fly from Madrid to Munich, and he will be interested in locating a service that can provide the booking of a seat on an appropriate flight. For locating services, consumers' objectives must be explicitly described, as presented in this section.

3.1 Description of effects

The first thing that must be described by a consumer is the set of effects he wants to achieve by using a service. In describing these effects, the same alternatives currently considered for the description of service capabilities are applicable: the consumer can describe the effects desired using natural language, giving a list of categories sought services must belong to, i.e., a coarse-grained view of the capability a service must have, or formally describing the set of effects desired and the intention of such description (*all* the effects in the set described are required or only *some* of them). For example, if a consumer wants to book a flight $flight1234$ (a flight from Madrid to Munich, operated by *air*, about which consumer has previously obtained information by using a flight search service), for passenger Ruben Lara, and paid with a DummyCard, the concept formalizing the set of effects required will be defined as (in terms of domain ontologies in $\mathcal{O}^{DL}$):

$Eff_{MyFlightBooking} \equiv Booking \sqcap ofItem.\{flight1234\} \sqcap forPerson.\{rubenLara\} \sqcap \exists withPaymentMethod \sqcap \forall withPaymentMethod.DummyCard$

The objectives of a consumer are modelled by a WSMO goal [10], as shown in Listing 2. This example goal describes the categories sought services must belong to (*category* non-functional property of the goal capability), the textual description of consumer's objectives (*description* non-functional property of the goal capability), and the formal set of effects expected with its corresponding intention (postcondition of the goal capability). The formal description of effects is restricted, as for services, to WSML-DL+, and in the example only some effect in this set is required, i.e., the booking of one seat with these characteristics is enough, as expressed by the value *some* of the associated intention.

3.2 Selection of filters

The second ingredient in the description of a goal is the level of accuracy expected from the results of the location process. In particular, the consumer

can choose to apply different filters to obtain services that can provide the effects required. These filters are split into two groups: 1) filters that can be applied by the registry where services are published (Section 4), without considering what input values can be provided by the consumer and, therefore, without evaluating information preconditions and input-dependent effects, and 2) filters whose application must be done once service descriptions are retrieved from the registry, i.e., at the consumer side and considering what information the consumer has available for providing input values to the service. Selected filters are encoded by the *filter* non-functional property of the goal capability. Details of the filters currently available in our model can be found in Sections 5 and 6.

```
goal _"http://www.afi.es/MyFlightBookingGoal"
  capability MyFlightBooking
   nonFunctionalProperties
     afi#category hasValue "http://www.afi.es/Taxonomy1#FlightBooking"
     dc: description hasValue "Booking of a seat on flight flight1234 from Madrid to Munich,
           paid with a DummyCard"
      afi#filter hasValue {"Capability", " InputAvailability "}
   endNonFunctionalProperties

   postcondition
     nonFunctionalProperties
       afi#intention hasValue "some"
     endNonFunctionalProperties
     definedBy
       ?x memberOf Eff_MyFlightBooking equivalent
            ?x memberOf Booking[ofitem hasValue flight1234, forPerson hasValue rubenLara] and
            exists ?cc(?x[withPaymentMethod hasValue ?cc]) and
            forall ?cc(?x[withPaymentMethod hasValue ?cc] implies ?cc memberOf DummyCard).
```

Listing 2: Excerpt of the description of an example goal

3.3 Input information

If filters involving the evaluation of possible input values for the service are selected (see Section 6), we will require a knowledge base KB to be defined. This knowledge base will contain the information the consumer knows and he is willing to disclose for achieving the described effects. This information will be defined in terms of domain ontologies $\mathcal{O}$ and thus, will consist of instances of WSML-Core concepts and properties.

KB will contain consumer knowledge such as user accounts, credit cards, personal details, etc., as well as other information such as the details of the flight $flight1234$ referenced in the formalization of effects in Listing 2. This knowledge base will be kept at the consumer's side, as it can be big in volume and it can include consumer's sensitive information e.g. credit card details which should not be disclosed to the registry or to any third-party at

location time [21]. In our current prototype, KB is implemented as a Flora-2[2] knowledge base containing consumer knowledge as instances of ontologies in $\mathcal{O}$. This knowledge base will be queried as described in Section 6.

In a nutshell, goals are defined by describing the effects desired, plus the filters that must be applied for finding relevant services. Additionally, if filters involving the evaluation of input values are selected, a knowledge base KB containing the information the consumer knows and which can be used to provide input values to relevant services must be in place.

4 Description and publication process

For services to be located they must be first described and their descriptions published so that potential consumers have access to them. In our model, we require the existence of a registry where service descriptions can be published and accessed by consumers and which, more specifically, can: a) process and store all the possible types of descriptions of a service considered, b) provide facilities for the location and retrieval of service descriptions based on the application of different filters over such descriptions, and c) define and manage reusable taxonomies of service categories.

We have designed and implemented a registry (see Figure 2), partly based on [30], which stores the descriptions of services discussed in Section 2 and allows for the application of filters over these descriptions and the descriptions of goals introduced in Section 3. This registry is conceived as an extension of a UDDI repository with improved service location capabilities based on the types of descriptions introduced in previous sections. Still, we allow for the direct usage of the UDDI API of the UDDI repository [6] in order to keep backwards compatibility with current service infrastructure and practices. In this way, consumers can choose between locating services directly using the UDDI API, and using the new location interface and the enhanced capabilities added on top of the UDDI repository.

In addition, the description of goals and services can be a challenging task for users. For this reason, we have incorporated service and goal description assistants, integrated in a service location client installed at the consumer side (see Figure 3), which interacts with the registry in order to support the description of services and goals by users.

In this section, we first present how taxonomies of categories are published and managed in our registry, as they have a prominent role in the categorization of services and in supporting users in describing services and

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

goals. Then, we describe the type of support offered to users in the description of services, and how service descriptions are published at our registry. Finally, we present how consumers are supported in describing their goals.

4.1 Taxonomies of categories

Our registry includes a taxonomy manager for the definition and management of taxonomies of categories, as shown in Figure 2. This manager allows for the following main operations, accessible as WSDL services:

Taxonomy creation. A new taxonomy must be given a URI for its identification. The manager stores it in the relational database used for persistency of the registry, creates a new UDDI tModel representing the taxonomy at the UDDI repository of the registry (jUDDI[3] has been used), and creates a TBox [26] in the DL reasoner of the registry (Racer [15] is used) whose name is the URI of the taxonomy.

Category creation. A name must be given to the new category, together with the URI of the taxonomy it belongs to, a textual description of the capability this category represents, a formal description of such capability and, optionally, a formal description of prototypical information preconditions and input-dependent effects associated to the category (see Section 2.3). All these elements will be stored in the relational database of the registry.

Previously to the storage of the elements above, the satisfiability of the concept formalizing the effects associated to the category e.g. concept $Eff_{FlightBooking}$ in Section 2.3 is checked. If the concept is satisfiable, it is published at the TBox whose name is the URI of the taxonomy the category belongs to e.g. *http://www.afi.es/Taxonomy1*, and the TBox is classified, i.e., the subsumption relation between the categories of the taxonomy, in terms of their associated formal capabilities, is computed[4]. The classified TBox will therefore correspond to a general-special hierarchy of the categories in the taxonomy, and this hierarchy will be stored in the relational database of the registry so that it can be later queried.

Checking satisfiability of concepts is NExpTime complete for the $\mathcal{SHOIN}$ Description Logic [24], as it is classification of a TBox, which can

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

[4] Strictly speaking, Racer only offers sound and complete reasoning for $\mathcal{SHIQ}$ and, therefore, nominals must be translated into pair-wise disjoint concepts before they are sent to the reasoner. Some incorrect inferences can be drawn from the resulting translation [16], but in most cases the subsumption and satisfiability relations computed will be correct. Recently, a sound and complete reasoning procedure for $\mathcal{SHOIN}$ has been implemented in the Pellet (http://www.mindswap.org/2003/pellet/) reasoner; testing the use of Pellet instead of Racer will be part of our future work.

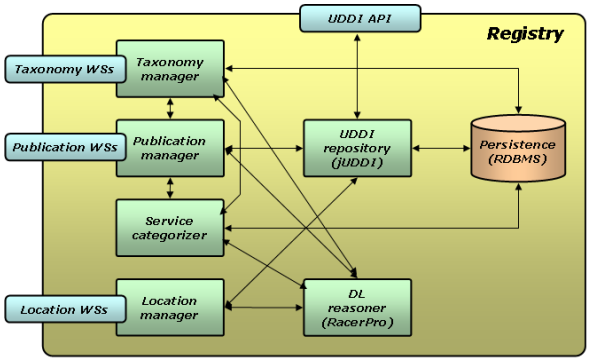

Figure 2: Registry architecture

be reduced to the reasoning task of checking satisfiability. Therefore, the times for creating a category can be high. Still, as this task is done off-line, i.e., without affecting service location times, it is not time-critical.

Category search. Two types of search of categories are supported: 1) given a textual description of a capability, and based on keyword matching, the categories whose associated textual description match the description given are returned, and 2) given a formal capability description in the form of a WSML-DL+ concept, the categories whose associated formalization of effects is equivalent, subsumed by, subsumes, or intersects the concept given are returned; this is done by querying the different TBoxes of the DL reasoner where categories are classified. Elements within each category are not ranked, but future extensions can incorporate a ranking based on e.g. [27].

4.2 Describing services

A provider will give a description of its service as a WSMO service, via the user interface of Figure 3, to a publication coordinator for its publication at the registry. Depending on the profile and skills of this provider, the WSMO service given will include certain types of descriptions of the service but not others e.g. will provide a categorization of the service but not a formalization of its capability. However, the more types of descriptions of the service are given, the more alternative matchmaking mechanisms (filters) we will be able to apply to locate this service (see Table 1). Therefore, if some types of descriptions of the service are not given the publication coordinator will, in cooperation with the service description assistant (see Figure 3), automatically propose to the user missing types of descriptions of his service based on the types of descriptions the provider has given.

In the following, we discuss the support mechanisms offered for the types

Applied filter	*Type of description of goal required*	*Type of description of service required*
Textual	Textual description	Textual description
Category	Categorization	Categorization
Capability	Formalization of effects	Formal capability
Input availability	Consumer KB	Information preconditions
Parameterized effects	Consumer KB + formalization of effects	Input-dependent effects

Table 1: Descriptions required for the application of different filters

of descriptions of services considered, which will be applied in the order presented.

4.2.1 Proposing categories

A provider can give a categorization of its service either manually or using the client user interface in Figure 3 to browse the taxonomies of categories available at the registry and select one or more of these categories. Still, if the WSMO service submitted to the publication coordinator does not contain the service categorization, we can propose categories the service might belong to by exploiting category search capabilities of the registry.

Proposal based on a formal capability. If a formal description of the service capability is given, but not its categorization, we can issue a category search request to the taxonomy manager to find and propose categories related to such capability. If we consider the service in Listing 1, the concept $Eff_{AirBooking}$ will be used by the taxonomy manager to query TBoxes storing taxonomies of categories for those categories whose associated formalization of effects is equivalent, subsumes, is subsumed by, or intersects this concept. The obtained categories have, thus, an associated set of effects related to the set of effects offered by the service, and they will be returned as the response to the search request, together with what particular relation they have to the service capability given.

The service provider is thereby proposed categories related in different ways to his service, and he can select one or more of these categories. For example, if the search request is issued with the capability formalized by concept $Eff_{AirBooking}$, the TBox defined for taxonomy *http://www.afi.es/Taxonomy1* will be queried and the *FlightBooking* category will be proposed to the provider as a category whose associated set of effects subsumes, i.e., is more general than the set formalized by $Eff_{AirBooking}$.

Proposal based on a textual description. If a textual description of the service is available, it can be sent to the taxonomy manager for a keyword-based category search. Returned categories are proposed to the service provider, who can choose to which ones his service will be assigned.

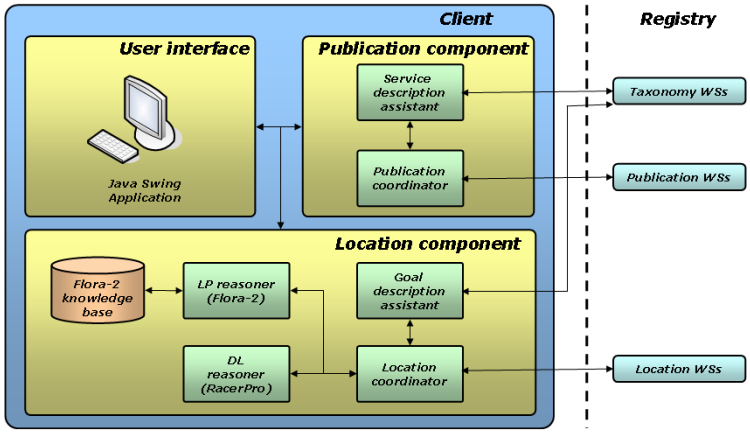

Figure 3: Client architecture

Summarizing, if the categorization of a service is not given by its provider, we will propose categories related to this service using both mechanisms above, if possible. However, if the proposal of categories based on the formal description of the service capability is feasible, we will recommend to the provider the selection of categories proposed in this way, as results of category search based on formal capabilities are more accurate.

4.2.2 Using categories to generate other descriptions

If the provider gives the categorization of his service (directly, by browsing available taxonomies, or by selecting some categories from the ones proposed as discussed above), we can use such categorization to propose missing types of descriptions of the service: we retrieve the textual description, the capability description, and the description of information preconditions and input-dependent effects associated to these categories, if available, and propose them for the missing types of descriptions of the service.

For example, let us imagine a provider gives a WSMO description of his service only including its categorization under category *FlightBooking*. In this case, we can propose the formal descriptions associated to the category (capability, information preconditions, and input-dependent effects), as well as the textual description of the category, for describing the service. The provider can later refine these proposed formal and textual descriptions, leading to e.g. the description in Listing 1.

4.3 Publishing services

A service provider, possibly after receiving the support of the service description assistant, will have a WSMO description of his service ready for publication, which might include certain types of descriptions but not others. In this section, we present how this description is stored and processed by our registry for its posterior retrieval.

4.3.1 Consistency of descriptions

When a WSMO service is submitted to the publication manager of the registry by the publication coordinator of the client, the first thing done by the publication manager is checking the consistency of formal descriptions of the service. For that purpose, the DL reasoner of the registry is used to evaluate the satisfiability of the concept defined as the conjunction of: a) the formal description of the service capability, and b) the union of the formal sets of effects associated to service categories. In this way, we are evaluating whether the sets of possible effects formalized by different types of descriptions are not contradictory, i.e., have a non-empty intersection. While checking concept satisfiability for WSML-DL+ ($\mathcal{SHOIN}$) is NExpTime-complete and, thus, time consuming, this task is not time critical.

4.3.2 Storing and processing descriptions

After checking consistency of the descriptions of a service $\mathcal{S}$, the publication manager will create a business service [6] at the UDDI repository. Furthermore, the WSMO description of the service and, separately, each type of description of the value of the service contained in the WSMO description, will be stored in the UDDI repository using tModels created for that purpose and associated to the business service created via keyed references [6]. Furthermore, the categories given for the service will be stored as the values of category bags of the business service [6] using the tModels created for the taxonomies these categories belong to, and the textual description of the service will be stored as the name of the business service. In this way, service descriptions will be directly accessible via the UDDI API.

Afterwards, the WSML-DL+ concept formalizing the set of effects associated to the service capability e.g. $Eff_{AirBooking}$ for the service in Listing 1, is sent to a TBox of the DL reasoner of the registry, named *Services*, and this concept is classified with respect to the TBox. Published services are thus arranged in a subsumption hierarchy in terms of the results they can provide, independently of input values. The classification of the TBox is done when

publishing the service as publication is not time critical, and this classification will reduce the times necessary to apply one of the filters described in Section 5. It must be noticed that the usage of a DL reasoner for classifying services in a subsumption hierarchy and later efficiently querying this hierarchy is possible thanks to the restriction in the expressivity of capability descriptions introduced in Section 2.

4.4 Describing goals

In order to locate appropriate services, consumers must explicitly describe the effects required from the usage of a service, the filters to be applied for its location and, in some cases, define a knowledge base containing the information available as possible input values for services. In general, a consumer will describe a WSMO goal including certain types of descriptions but possibly not others, as well as the filters to be applied for goal resolution, and will possibly inform of the location of a knowledge base containing the information he has available.

Different filters require different types of descriptions of the consumer goal and of published services for their application, and some also require a consumer knowledge base to be defined. For this reason, when the consumer goal is submitted to the location coordinator of the location component (see Figure 3) for its resolution, this coordinator will first check whether the necessary types of descriptions of the consumer objectives have been given for the application of the selected filters. The relation between selected filters and required descriptions is given in Table 1.

If the types of goal descriptions required for the application of selected filters are not available, the goal description assistant of the location component will warn the user and automatically propose missing types of descriptions. In particular, we can propose categories starting from a textual description or from a formal description of the effects required, in the same way categories were proposed for the categorization of services (see Section 4.2). Furthermore, we can use the descriptions associated to categories given with the goal to propose a textual description and a formal description of consumer's objectives.

As we can see, if some type of description of consumer's objectives, necessary for the application of the selected filters, is not available, we will support the consumer in providing it by proposing the missing type of description. Still, the consumer can modify the proposed descriptions in order to make them more accurate. Additionally, if the consumer knowledge base has not been given and it is necessary for the application of the filters selected, the consumer will be required to give its location.

It must be noted, though, that we only propose the types of descriptions of a goal necessary for the application of selected filters, not missing information for completely matching candidate services in the way proposed in [8].

5 Registry-side Filters

When the consumer submits his goal for resolution to the location coordinator of Figure 3, and if the necessary types of descriptions for the application of the selected filters are available, the coordinator will first send this goal to the location manager of the registry. This manager will apply the registry filter selected by the user, and relevant services will be retrieved from the registry; currently, one of three alternative filters can be applied for retrieving services from the registry, namely: a textual filter, a category filter, or a capability filter. From these, the least expensive filter but also the least accurate will be the textual filter; the capability filter, which exploits formal descriptions of the set of effects associated to the service capability, will be the most expensive and also the most accurate one.

5.1 Textual filter

If the textual filter has been selected, indicated by the *textual* value of the *filter* property of the goal, a simple keyword-based matching of the textual description of consumer objectives against the textual descriptions of services published at the registry will be performed, and services matching some of the keywords in the goal textual description will pass this filter.

The application of this filter will provide results in very short times, as only a simple query over the registry database will suffice to find relevant services. Furthermore, the filter only requires a type of description of services and goals which is easy to provide by users. However, the precision of the results provided is limited, and we currently do not distinguish different levels of match (all services that pass the filter are considered a perfect match).

5.2 Category filter

If the category filter is selected, the location manager extracts the categories specified by the *category* non-functional property, which are the categories sought services must belong to (interpreted as a logical *and*). The hierarchy of the taxonomies these categories belong to, stored in the registry database (see Section 4) and of which a copy is kept in memory for fast access, will be queried for: a) categories more general (parents), and b) categories more

specific (children) of the categories given. We will consider a service to be a perfect match if it has been assigned to all the categories given by the goal or to parents of these categories. Services not fulfilling this condition but belonging to some category given by the goal, or to a parent or children of such a category, will be considered a partial match.

If we consider the goal in Listing 2, services belonging to category *FlightBooking* or to more general categories e.g. *TransportBooking* at taxonomy *http://www.afi.es/Taxonomy1* will be considered a perfect match, and services belonging to more specific categories e.g. *LowCostFlightBooking* will be considered a partial match.

In order to apply the filter as explained above, only simple lookups over a copy of the taxonomies published, which is kept loaded in memory together with a relation of what services are assigned to what categories in such taxonomies, will be done and, thus, response times obtained are low. 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 the categories defined.

5.3 Capability filter

If the capability filter is selected, the concept formalizing the set of effects required by the consumer will be used for matching services whose set-based modelling of capability effects relates in some way to this concept. However, before the goal is submitted to the registry, instances not in domain ontologies $\mathcal{O}^{DL}$ must have been substituted by their definitions in terms of concepts, relations and instances in domain ontologies, as otherwise the description will not be processable by the registry.

For example, if we consider the goal in Listing 2, instances $flight1234$ and $rubenLara$ must be replaced by their definitions in the formalization of the set of effects required:

$Eff_{MyFlightBooking} \equiv Booking \sqcap \exists ofItem \sqcap \forall ofItem.(Flight \sqcap hasOrigin.\{madrid\} \sqcap hasDestination.\{munich\} \sqcap operatedBy.\{air\}) \sqcap \exists forPerson \sqcap \forall forPerson.(Person \sqcap countryOfResidence.\{spain\}) \sqcap \exists withPaymentMethod \sqcap \forall withPaymentMethod.DummyCard$

For a goal $\mathcal{G}$, the location manager will query the *Services* TBox of the DL reasoner of the registry for concepts: a) equivalent to, b) more general than, c) more specific than, and d) with a non-empty intersection with concept $Eff_{\mathcal{G}}$. Therefore, concepts defining a set of effects related in some way to the effects required are located. Once these concepts are retrieved, we will use the criteria in [18] to determine, taking into account intentions and the relation between the set defined by these concepts and the set of effects

required by the goal, as shown in Table 2, which of them are a perfect, possible perfect, partial, or possible partial match for the goal.

In Table 2, $I_{\mathcal{S}}$ denotes the intention of the service capability, $I_{\mathcal{G}}$ the intention of the set of effects formalized by the goal, $Eff_{\mathcal{S}}$ the formalized capability of the service, and $Eff_{\mathcal{G}}$ the formalized set of effects expected by the consumer. Notice that the set-theoretic relations used in the table correspond to the subsumption relations obtained between the WSML-DL+ concepts formalizing the sets the table refers to.

A perfect match (Match) means that the effects required by the consumer can be provided by the service. For example, if the consumer only requires some effects from the set formalized ($I_{\mathcal{G}} = some$), the service offers all the effects in its formalized capability ($I_{\mathcal{S}} = all$), and the sets have a non-empty intersection ($Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} \neq \emptyset$), we can be sure that the service can offer some of the effects in the set described by the consumer and, thus, completely satisfy the consumer's needs. A partial match (PMatch) means that only part of the effects required by the consumer are offered by the service. A possible match (poMatch) means that there might be a perfect match, but we cannot guarantee it. For example, if we have that the service offers only some of the effects in the set formalized by its capability ($I_{\mathcal{S}} = some$), the consumer requires all the effects described ($I_{\mathcal{G}} = all$), and the relation between sets is $Eff_{\mathcal{G}} \subset Eff_{\mathcal{S}}$, it might be the case that the effects in the set $Eff_{\mathcal{S}}$ provided by the service are precisely those in the set $Eff_{\mathcal{G}}$, but we have no guarantee of it and it can turn out to be a partial match or a non-match. A possible partial match (ppMatch) refers to the situation where there might be a partial match between the set of effects required by the goal and offered by the service, but we have no guarantee of it and it can turn out to be a non-match. Finally, a non-match (NoMatch) refers to cases in which the service does not offer any of the effects required by the goal, i.e., in which the service cannot contribute at all to solve the consumer's goal. The application of the filter will, therefore, result in four sets of matching services, one per type of match.

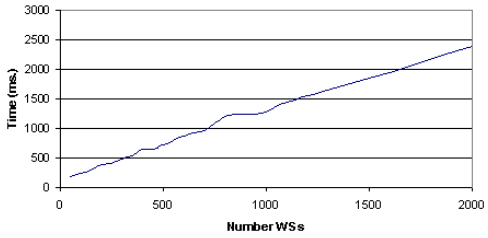

Figure 4: Times for the application of the capability filter

Intention of $\mathcal{G}$ / $\mathcal{S}$	$I_{\mathcal{S}} = all$		$I_{\mathcal{S}} = some$	
$I_{\mathcal{G}} = all$	$Eff_{\mathcal{G}} = Eff_{\mathcal{S}}$	Match	$Eff_{\mathcal{G}} = Eff_{\mathcal{S}}$	PMatch
	$Eff_{\mathcal{G}} \subset Eff_{\mathcal{S}}$	Match	$Eff_{\mathcal{G}} \subset Eff_{\mathcal{S}}$	poMatch
	$Eff_{\mathcal{G}} \supset Eff_{\mathcal{S}}$	PMatch	$Eff_{\mathcal{G}} \supset Eff_{\mathcal{S}}$	PMatch
	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} \neq \emptyset$	PMatch	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} \neq \emptyset$	ppMatch
	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} = \emptyset$	NoMatch	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} = \emptyset$	NoMatch
$I_{\mathcal{G}} = some$	$Eff_{\mathcal{G}} = Eff_{\mathcal{S}}$	Match	$Eff_{\mathcal{G}} = Eff_{\mathcal{S}}$	Match
	$Eff_{\mathcal{G}} \subset Eff_{\mathcal{S}}$	Match	$Eff_{\mathcal{G}} \subset Eff_{\mathcal{S}}$	poMatch
	$Eff_{\mathcal{G}} \supset Eff_{\mathcal{S}}$	Match	$Eff_{\mathcal{G}} \supset Eff_{\mathcal{S}}$	Match
	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} \neq \emptyset$	Match	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} \neq \emptyset$	poMatch
	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} = \emptyset$	NoMatch	$Eff_{\mathcal{G}} \cap Eff_{\mathcal{S}} = \emptyset$	NoMatch

Table 2: Degree of match taking into account intentions

As the TBox queried for retrieving matching services has been already classified when publishing services, and as the definition of service capabilities and required effects is restricted to the $\mathcal{SHOIN}$ DL, existing DL reasoners can provide responses to the queries over the TBox in relatively low times. For estimating response times, we have generated 2000 random variations of the capability of the service in Listing 1, describing the capability of booking flights operated by different airlines, for different itineraries (restricted to certain cities, countries or continents), and with different payment methods, and we have also introduced services that offer different actions not related to flight booking. The times measured, using a computer with an Intel Pentium 4 2.8GHz processor and 1GB RAM, are shown in Figure 4 as a function of the number of services published. A remark is in place: querying Racer for equivalent, more general or more specific concepts yields response times below 20 milliseconds for 2000 services published (see [21]), while querying for non-intersecting concepts (in order to afterwards obtain intersecting concepts) is the most time-consuming operation. The reason is that the algorithms used by DL reasoners to classify the TBox are optimized to compute hierarchical relations and they do not pre-compute disjoint classes.

The application of this filter will be considerably more time-consuming than the other filters previously presented. Furthermore, it relies on formal descriptions which are more difficult to provide by users, although this difficulty is reduced by the use of the description assistants presented. However, the description of relevant sets of effects can be much more accurate than the textual description or the categorization of a service or goal, and the results

obtained from the application of this filter will be more accurate.

If we compare the notions of match used by works which base the matching of services and goals on DL reasoning, in [28] and [22] the following notions are employed: a) exact, corresponding to equivalent concepts, b) plug-in, corresponding to capabilities more general than (subsuming) goals, c) subsumes, corresponding to capabilities more specific than (subsumed by) goals, d) intersection (only considered in [22]), corresponding to capabilities and goals with a non-empty intersection, and e) failed matches, corresponding to non-intersecting concepts. Both works focus on the matching of OWL-S inputs and outputs, and they implicitly assume a universal intention in the descriptions used. Given this assumption, we can establish a rough mapping of the degrees of match used by our capability filter to those proposed by these works: an exact match and a plug-in match can be mapped to a perfect match in our model, a subsumes match to a partial match, and a failed match to a non-match only when both concepts do not intersect; otherwise, we consider this case a partial match. In [27], three notions of match are distinguished: a) an exact match, which corresponds to equivalent concepts, b) a potential match, meaning that some aspects of the request are not specified in the offer, but the concepts intersect, and c) a partial match, meaning that some aspects of the request are in conflict with the offer (non-intersecting concepts). In general, an exact match as defined in [27] corresponds to a perfect match in our model independently of the intentions associated; a potential match can correspond to a perfect match, to a possible match, to a partial match, or to a possible partial match in our model depending on the intentions associated to the concepts evaluated and on the particular subsumption relation which may hold; and a partial match corresponds to a non-match in our setting. Still, none of these works allows for the explicit association of intentions to the DL concepts used to model offers and needs.

After applying one of the filters above, chosen by the consumer, the complete WSMO descriptions of matched services will be obtained from the UDDI repository and returned to the location client.

6 Consumer-side Filters

After applying a registry-side filter, the description of services that can fulfill the consumer goal are retrieved from the registry and returned to the location coordinator. In particular, four sets of service descriptions are retrieved, corresponding to the different notions of match discussed in Section 5 (remember that, if the textual filter is applied, all matched services will be perfect matches and, if the category filter is applied, only perfect and partial

matches are distinguished). If no consumer-side filter is selected, the location coordinator directly provides these service descriptions to the consumer. Otherwise, the two consumer-side filters currently available can be successively applied in order to narrow down the set of relevant services obtained, namely: an input availability filter and a parameterized effects filter.

6.1 Input availability filter

Besides checking whether a service is relevant for solving a consumer's goal in terms of its capability, the consumer might want to know whether the information preconditions of the service can be fulfilled, i.e., whether the consumer has appropriate information available to be submitted to the service as input values.

For this purpose, we will use the information preconditions of services retrieved from the registry as queries to the knowledge base KB containing consumer knowledge, i.e., we query for valid input values for the service. Preconditions are expressed as WSML-Flight queries (see Section 2.2) and, thus, they have LP semantics. As discussed in [21], LP semantics are appropriate in this setting as we want to determine whether valid input values can be provided to the service *from knowledge base* KB (closed world), i.e., whether the consumer has available appropriate information to be provided as input values to the service. Furthermore, for obtaining valid input values we are only interested in efficient query answering, not in general entailment.

As an example, let us consider the goal in Listing 2. After submitting the goal to the registry, the service *AirBooking* described in Listing 1 will be retrieved as a perfect match for the goal as, according to its capability formalization, it can provide the flight operated by airline *air* required for the passenger indicated. We will then evaluate at the consumer side, once the description of the service is retrieved from the registry, whether the consumer has appropriate information in his knowledge base to be provided as input values to the service.

Let us imagine knowledge base KB contains two instances $myCC$ and $myCC2$ of $DummyCard$, corresponding to consumer's credit cards, as well as instances $rubenLara$ and $flight1234$, containing the details of the passenger and the flight operated by *air* for which a booking is to be made, respectively. We can use the formalization of information preconditions of the retrieved service as a query to KB, obtaining two possible valid input bindings: $\beta_1 = \{f = flight1234, p = rubenLara, cc = myCC\}$ and $\beta_2 = \{f = flight1234, p = rubenLara, cc = myCC2\}$. Therefore, we can conclude that the consumer can provide valid input bindings for using the service and, thus, this service will pass the filter.

Input availability will act a boolean filter, i.e., the information requirements of a service will be fulfilled or not. However, as we allow providers to publish services with different level of detail in their description, there might be services retrieved from the registry which do not describe their information preconditions. Therefore, the filter will yield: 1) services that passed the filter, and 2) services to which the filter could not be applied. Services will be grouped into these categories and, if no other filter has been selected, their complete WSMO descriptions will be returned to the consumer along with their degree of match given by the application of registry-side filters.

It must be noted that query answering for WSML-Flight can be done in polynomial time [9]. In our prototype, the knowledge base KB, implemented as a Flora-2 knowledge base, is defined before any goal is issued to the service location component. Therefore, it is already compiled and loaded when a goal has to be resolved, which makes querying for valid input values using the definition of information preconditions of the service efficient (less than 50 milliseconds for a knowledge base with more than 14000 randomly generated facts, according to out tests).

6.2 Parameterized effects filter

As discussed in Section 2, there is a dependency between service effects and the input values provided by the consumer. If the parameterized effects filter is selected, we will not only check whether the customer has available appropriate information for providing valid input bindings for services retrieved from the registry, but also how the set of effects that can be obtained depends on such bindings.

In particular, after the application of the input availability filter we will obtain, for each service $\mathcal{S}$ which passed the filter, a set $\Sigma = \{\beta_1, \ldots, \beta_n\}$ of valid input bindings for the service. Each binding β_i from this set can be used to execute this service, leading to the obtention of a certain set of effects $Eff_{\mathcal{S},\beta_i}$ which is defined by substituting input variables by the corresponding values in the input-dependent formalization of effects of the service (see Section 2.2).

If the consumer would execute the service for each valid input binding he can provide, the set of effects that could be obtained is defined by the union of the sets of effects obtainable for each input binding, i.e., by the union $Eff_{\mathcal{S},\Sigma} \equiv \bigsqcup_{\forall \beta_i \in \Sigma} Eff_{\mathcal{S},\beta_i}$

If we continue with the example above, in which the set of input bindings $\Sigma = \{\beta_1, \beta_2\}$ was obtained for service $AirBooking$, the set of effects that can be achieved by using the service with these input bindings is defined by:

$InputEff_{AirBooking,\Sigma} \equiv$

$(Booking \sqcap ofItem.\{flight1234\} \sqcap forPerson.\{rubenLara\}) \sqcap withPaymentMethod.\{myCC\}) \sqcup$
$(Booking \sqcap ofItem.\{flight1234\} \sqcap forPerson.\{rubenLara\}) \sqcap withPaymentMethod.\{myCC2\})$

The concepts obtained in this way for each relevant service retrieved from the registry will be published at the TBox of the DL reasoner used by the service location component (Figure 3). The location coordinator will then query for concepts: a) equivalent to, b) more general than, c) more specific than, and d) intersecting the formalization of the set of effects expected by the goal. Table 2 will be again applied, yielding the list of 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 services that offer, *for the input values available*, the effects expected by the consumer, considering how these effects depend on the input values provided.

As some services might not describe input-dependent effects, the service location coordinator returns to the consumer: 1) services that passed the parameterized effects filter, 2) those that passed the input availability filter but to which the parameterized effects filter could not be applied, and 3) those to which none of the filters could be applied. Services will be grouped into these categories, and their complete descriptions will be returned to the consumer along with their degree of match.

Response times for the application of this filter are expected to be relatively high. The reason is that the publication, for its posterior querying, at the TBox of the DL resoner of concepts $Eff_{\mathcal{S},\Sigma}$ which model the set of effects obtainable from each relevant service $\mathcal{S}$ for particular input bindings Σ is done at run-time. Thus, the TBox of the DL Reasoner cannot be fully classified before-hand in order to reduce the response times of querying this TBox. While we expect most services to be discarded by previous filters so that not many services have to be evaluated in this way, the times required are still high e.g. around 60 seconds for 500 services evaluated.

By applying consumer-side filters, we can evaluate whether there are input values available satisfying information preconditions, as well as whether the effects obtainable for such values can achieve the consumer's goal. However, real world preconditions and their influence on service effects are not considered. The reason, as explained in [21], is that the location process will generally not have access to the information required to evaluate preconditions. For example, let us imagine our flight booking service requires the availability of a free seat on the flight given. The fulfillment of this real world precondition might not be directly checkable by the location process, as it is information generally known by the service provider. Therefore, for the evaluation of this condition, the location coordinator would have to locate

a service that informs of the availability on the required flight, and would have to execute it. Furthermore, this service could in turn require the evaluation of some additional real world preconditions. Therefore, we would have a situation in which the recursive location and execution of services might be required for finding a service that can solve a consumer goal, which is not desirable.

7 Related Work

Most service location proposals are based on the semantic matchmaking of services and goals based on DL subsumption reasoning e.g. [3, 22, 28] focusing on a single logical filter and not considering the usage of alternative filters. In general, these works only describe the type of input values expected and the type of effects offered by services, without describing how they relate. Goals describe the type of input values that can be required by available services and what effects must be offered by these services. These descriptions of inputs and effects, given by services and goals, are compared in order to find services usable to achieve the goal. However, we believe this treatment of inputs is closer to a signature matching than to a matchmaking based on the functionality of services.

Furthermore, consumers are expected to explicitly describe in advance, as part of their goals, *the type* of input values they can provide for achieving the effects desired. This requires consumers to anticipate what type of input values might be required by relevant services. On the contrary, we believe in most cases consumers will want to find services *which can provide the effects required and which can be executed*, i.e., the primary concern of a consumer is to obtain some effects from the execution of a service, which requires having appropriate input values for this execution; consumers will not require a fixed input signature from the service, as long as they can execute the service with the information they have available. For this reason, we have introduced our input availability and parameterized effects filters, which enable a type of matchmaking different from what existing works propose. Still, the matching of the input signature of services might be interesting for location of services at design-time and their composition or integration into more complex systems or processes, where a service not only providing a certain capability and functionality, but also with a given signature, might be required. For this reason, we envision the extension of our model to include this type of signature matching, which will require the description, as part of the goal, of the type of inputs that can be required by services.

On the other hand, the works presented in [17] and [19] take into account

how the effects of executing a service depend on the input values provided to it. The former makes use of Transaction Logic [4] and Logic Programming, which presents some problems with the treatment of unspecified information in goals and service capability descriptions. The latter, based on Description Logics, expects customers to explicitly describe what kind of relation they expect between effects and inputs, which we believe is a less usable approach than that taken by our model.

The only approaches we are aware of that apply different types of filters are LARKS [31] (for the matchmaking of agents) and OWLS-MX [20] (for the matchmaking of OWL-S 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 only work over descriptions with first-order semantics, b) whether the customer can actually provide appropriate input values is not evaluated, c) how particular input values affect the set of effects that can be obtained is not considered, d) all matchmaking is done at the registry side, and e) customers are not supported in describing goals and services. However, it must be noted that filters based on the matching of textual descriptions described by [20] (or [12]), could be incorporated to our model as registry-side filters in order to offer more fine-grained results than those currently provided by our simple keyword-based filter.

In a nutshell, we are not aware of any work on the location of services with all the features of our model and prototype, especially with respect to the consideration of alternative types of descriptions of services and goals, the offer of filters with different trade-offs between accuracy and efficiency, the assistance to users in describing their services and goals, and the treatment of input values and how effects of the service depend on such values.

8 Conclusions

The increase in the number of services defined, as a consequence of the growing adoption of the SOA paradigm, requires appropriate facilities for the location of services with a certain value. Furthermore, a model for the location of services must be flexible in the kind of descriptions of artifacts expected, as well as in the matchmaking mechanisms users can apply for locating services that can solve a given goal. Last, but not least, the task of describing services and goals must be supported so that users can face it with guarantees.

For all these reasons, we have have elaborated a model and implemented a prototype that accommodates different types of descriptions of services and goals, enables the selection of filters with different properties for obtaining

services so that results can be obtained with different trade-offs between accuracy and efficiency, and incorporates assistants for the description of services and goals. This results in a model and prototype which are flexible and usable in different scenarios.

Future work will focus on the optimization of the prototype and on the incorporation of new filters to the set of filters currently designed and implemented. Envisioned optimizations include the possible adoption of other reasoners, such as Pellet, in our registry and location client, as well as the automation of certain tasks which are currently done manually in the prototype, such as the substitution, in the formalization of required effects, of instances which are not in domain ontologies by their definitions in terms of these ontologies.

References

[1] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet. Semantic Web Services Framework (SWSF) overview. Technical report, W3C submission, September 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] B. Benatallah, M. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *VLDB*, 14:84–96, 2005.

[4] A. J. Bonner and M. Kifer. Transaction Logic Programming (or, A Logic of Procedural and Declarative Knowledge). Technical report, University of Toronto, November 1995.

[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] L. Clement, A. Hately, C. von Riegen, and T. Rogers (editors). UDDI Version 3.0.2, october 2004.

[7] OWL-S Services Coalition. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1, November 2004.

[8] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. Concept abduction and contraction for semantic-based discovery of

matches and negotiation spaces in a E-marketplace. *Electronic Commerce Research and Applications*, 4(4):345–361, 2005.

[9] 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.

[10] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kiffer, B. Koenig-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg. Web Service Modeling Ontology (WSMO). Submission, W3C, June 2005.

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

[12] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *VLDB 2004*, 2004.

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

[14] B.N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *WWW'03*, 2003.

[15] Volker Haarslev and Ralf Möller. RACER System Description. volume 2083, 2001.

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

[17] D. Hull, E. Zolin, A. Bovykin, I. Horrocks, U. Sattler, and R. Stevens. Deciding semantic matching of stateless services. In *AAAI-06*, 2006.

[18] U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *ESWC 2005*, Heraklion, Greece, May 2005.

[19] 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.

[20] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *AAMAS 2006*, 2006.

[21] R. Lara. Two-phased web service discovery. In *AI-driven Service Oriented Computing workshop at AAAI 2006*, July 2006.

[22] L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *WWW'03*, Budapest, Hungary, May 2003.

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

[24] C. Lutz. An improved nexptime-hardness result for description logic alc extended with inverse roles, nominals and countins. Technical report, Technical University Dresden, 2004.

[25] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz (eds.). Reference Model for Service Oriented Architecture 1.0. Technical report, OASIS, 2006.

[26] D. Nardi, F. Baader, D. Calvanese, D. L. McGuinness, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge, January 2003.

[27] T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mogiello. A system for principled matchmaking in an electronic marketplace. *International Journal of Electronic Commerce*, 8(4):9–37, 2004.

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

[29] 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.

[30] N. Srinivasan, M. Paolucci, and K. Sycara. Adding OWL-S to UDDI, implementation and throughput. In *SWSWPC Workshop at ICWS 2004*, 2004.

[31] K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *AAMAS 2002*, 2002.