WSML Deliverable

D5.1 v0.1
WSMO Discovery

WSML Working Draft – September 13, 2004

Authors:
Uwe Keller, Rubén Lara, Axel Polleres, Ioan Toma, Michael Kifer, and Dieter Fensel

Editors:
Uwe Keller, Rubén Lara, and Axel Polleres

This version:
http://www.wsmo.org/2004/d5/d5.1/v0.1/20040906
Latest version:
http://www.wsmo.org/2004/d5/d5.1/v0.1/
Previous version:
http://www.wsmo.org/2004/d5/d5.1/v0.1/20040802

Abstract

The *Web Service Modeling Ontology* (WSMO) provides the conceptual framework for semantically describing web services and their specific properties. Based on WSMO, the *Web Service Modeling Language* (WSML) implements this conceptual framework in a formal language for annotating web services with semantic information.

In this document we present and discuss how (semantic) descriptions of web services based on WSMO and WSML can be exploited for the dynamic discovery i.e. location of suitable services to fulfill a given objective.

In order to cover the complete range of scenarios that can appear in practical applications, several approaches to achieve the automation of the discovery process are presented and discussed. They require different levels of semantics in the description of web services and requests, and have different complexity and precision.

WSMO, WSML, and the approaches for discovery presented in this paper constitute the necessary infrastructure to achieve the automatic location of web services in different scenarios, ranging from natural language descriptions of web services and requests to precise logical definitions of the web services functionality and the requester objectives.

Contents

1 Introduction

1.1 Motivation

The World Wide Web (WWW) has been invented in 1989 by Tim Berners-Lee [BLFD99] and since then changed the way people gather and access knowledge in everyday life. At its invention, nobody was able to foresee the effects and influences of this new promising technology. In fact, the WWW can be considered as one of the most influential technologies that has been invented in the 20th century.

Nowadays, the World Wide Web is certainly the biggest information repository, containing a huge amount of knowledge. But the major bottleneck when it comes to exploiting this information is how to find the required knowledge: the Web is still a mainly syntactically-driven information repository, where information is represented in form of web pages that actually fail to capture the semantics of their contents. As a consequence, the content of the pages is *not* easily accessible by computers but instead exclusively intended for human consumption. Given the size and the rapid growth of the WWW, this approach does not scale up.

This has led to significant research on the so-called *Semantic Web* [BLHL01, DFvH03, FHLW03], that aims at enabling computerized agents to easily gather information in the Web and exploit this knowledge on behalf of their human clients by enriching information in the WWW with machine-processable semantics.

From Web Services to Semantic Web Services. Web services add a new level of functionality on the current Web. Whereas at the moment the Web is mainly constituted by simple documents (static and dynamic web pages), in the future the Web might become a repository of electronic (web) services. Each service represents a computational entity that can be accessed and invoked over the Internet. Thus, the Web might become a huge, distributed and shared computational device.

Recently, web services have gained more and more interest in industry as well as among academics. This is due to what they promise: a uniform infrastructure for accessing software-systems over the Internet, which can be used for integration of legacy systems, even across enterprise boundaries. Nonetheless, though web service simplify enterprise application integration, the technologies around web services – namely SOAP [W3C03], WSDL [CCMW01] and UDDI [BCE$^+$02] – do not solve the integration problem by themselves. As the Web itself, current integration technologies are exclusively syntactical means that lack explicit, machine-understandable semantics.

Moreover, given the growing industrial interest in web services, the development and publication of a huge amount of web services for solving various tasks can be expected in the near future. Hence, the same bottleneck as for the conventional WWW arises: How can we find a web service that solves a problem at hand? SOAP, WSDL and UDDI are not sufficient for answering this question. Again, machine-processable semantics for web services is needed in order to ensure access of web services for agents without human interaction. Adding semantics to web services is the main prerequisite for an approach to scale.

This semantically enriched web services are called *Semantic Web Services*. In a sense, we can use the following equation to summarize the relationship between web services and semantic web services:

Semantic Web Service = Web Service + Semantic Annotation

And what do we do with all that Semantics? The major benefit of any form of specification is that by explicitly documenting what an artifact actually does, it becomes a lot easier for humans (who do not know that artifact before) to comprehend what a given artifact can be actually used for.

Clearly, with a formal representation of any specification, the purpose and use of an artifact becomes more precise and eventually accessible for *computerized* agents. Knowledge about the capabilities of single objects can be exploited for combining these objects to systems which solve more complex problems than each of their elements themselves.

Each software system is build for some purpose: to solve a specific well-defined problem of a client or organization. Software architectures themselves are the conceptual means for describing how the functionality of a system is provided and how the given requirements are actually satisfied. In this sense software architectures [GS96, BCK03] bridge the gap between the user requirements and the level of computational units that provide some well-defined functionality and interact and communicated with each other.

In a world of rapidly changing requirements and incomplete knowledge (such as the Semantic Web with its open and heterogeneous user community), the problems to be solved most likely occur rather ad-hoc and cannot be foreseen in detail. In this case, the complete construction of a system that solves the problem at hand, in general cannot be done in advance, but has to be done on the fly instead. Thus, software-systems composed by services in the Web constitute a particularly interesting and challenging problem for the theory and practice of software architecture: how to dynamically construct systems out of existing ones in a goal-driven way.

For this purpose, computerized agents as well as human software engineers need support for *reasoning about semantic web service descriptions* in order to effectively utilize semantic annotations for the system creation task and its subtasks: discovery, composition, configuration, execution and monitoring of adequate computational components to build a required software system in a dynamic way.

1.2 A comprehensive Overview on Semantic Web Service Discovery

In this document we focus on one of the afore mentioned tasks in the construction process, namely *discovery* of suitable computational components or services, respectively.

What is Service Discovery? We want to start with an informal definition of the notion *service discovery* in order to achieve a common understanding with the reader. This interpretation of the notion will be used throughout the whole document. Later on, we will provide an abstract model of service discovery in WSMO and derive a precise (and partly formal definition) of service discovery in WSMO.

Informally, we consider *service discovery* as the process of identifying possible candidates of services that can solve a client's problem. According to this definition the notion is inherently *relative* to several parameters:

(a) Some *base set of services* which has to be considered for the processing of a discovery request, for instance the set of all services (or more precisely, the set of all *known* services, where ,,known" could refer to being registered

in some (specific) service registry or a set of service registries, or some arbitrary fixed set of services which will be defined just before to the processing of the discovery request starts[1].

(b) Some *specification of the characteristics of the services* to be searched for.

These specifications can include various kinds of information and apply several different techniques for expressing this information, ranging from simple keywords to precise formal descriptions of service properties:

Since it is reasonable to assume that someone is looking for service because he eventually wants to use[2] some (adequate) service in order to achieve a well-defined goal, the most fundamental and important part of every service description is concerned with the *functionality* provided by the services. In WSMO, this part of a service description is called *capability* of the service.

In regard to the service discovery, WSMO follows a goal-driven approach where the user does not directly specify what functionality he is looking for, but rather specifies the state of the world he wants to achieve with the execution of some adequate service. Thus, a client specifies the required service functionality only indirectly by means of the observable ,,effects" of a service execution.

Of course, besides the functionality of a service, there are other characteristics as well which are relevant for a client and thus should be included in the search request, most notably required *non-functional properties* of a service (e.g. response times, levels of trust, availability etc.) and possible user preferences.

(c) A *definition of what services are considered as relevant* wrt. to some discovery request. This definition has to be translatable into some algorithm which exploits the available information. Relevance does not need to be a crisp criterion (i.e does not need to judge true or false) but in general is considered to be some fuzzy criterion, which takes discrete values in some ordered domain.

In other word, we need to have a formal model for matching and degrees of matching. The result of the discovery process can be decreasingly ordered according to the degree of matching (but does not have to be).

For this purpose we consider two formal components in our discussion of Service Discovery in this document: A *matching notion* (which identifies relevant services for some service request) and a *ranking function* (which determines the degree of relevance of the identified candidates).

An abstract Perspective on Service Discovery. *Some of the abstract mathematical model of Discovery from older versions!*

Aspects of Service Discovery. Let us briefly overview all the aspects of service discovery which are relevant when it comes to an actual realization of such a system in form of a software. Figure 1 illustrates and summarizes our discussion in this paragraph. In our opinion, all the mentioned aspects have to be investigated and the related problems have to be solved adequately in order to build a useful and industrial-strength service retrieval system.

[1] The latter situation in particular occurs when one considers discovery as a levelled *filtering process* which happens in stages whereby the discovery component consists of a sequence or a partial-order of filters which address particular aspects of a discovery request. We will come back on this issue later on in this document.

[2] Technically, this means to execute a service.

Service discovery is some specific form of retrieval task where the search request as well as the items to be retrieved have particular properties and capture relevant information in a specific manner. The approach to service discovery that is to be proposed in this document clearly shares some commonalities with classical Information Retrieval. Of course (and even luckily), there are a quite a few differences as well. Getting an understanding about the relationship between Service Discovery and Information Retrieval potentially opens the possibility to find, transfer and reuse or adapt already well-understood techniques and perhaps even existing systems from an likewise old and well-investigated discipline. For this reason, we believe that it makes perfect sense, to analyze and clarify this relationship.

Our final goal is to build a concrete system which realizes Web Service Discovery based on the WSMO methodology. For any such system, it is needed to develop a conceptual model of how the discovery process basically works, which (conceptual) components are involved and what aspect of the process these components actually address and realize. We call this conceptual model an *abstract Architecture of Service Discovery*. In general, we can distinguish components which are based on information about the single items that can be retrieved (and the description of the items that have to be retrieved) without explicit, formal semantics and component which actually take semantics of formal descriptions into account. The results of the former more or less implement some sort of heuristics[3] for identifying suitable candidates whereas the latter components perform some processing of available *explicit semantic descriptions* of the potential candidates and thus in general do not represent heuristics but directed, tailored methods which can provide rather accurate results. Thus, the first type of components will be called *syntax-based components*, whereas the latter type of components will be called *semantics-based*.

For an actual approach, one has to define and design the single components to be used for discovery. This includes to get a clear understanding of what problem they address and solve and how this is achieved in detail.

Then, the single components have to be combined into an architecture which realizes the overall discovery process. The interaction and interplay between the single (conceptual) components has to be specified.

The abstract architecture has to be justified with respect to requirements and assumptions that one expects for the concrete system. In particular, the main design decision should be explained and the available options should be discussed.

Next, using this architecture and its components one has to explain and fix the precise semantics of the discovery process and its outcome. That means one has to precisely specify how the result computed and how it can be interpreted with respect to the information that has been put into the overall process. In other words, we define the formal meaning of the model. In particular, this involves to formally define how to detect possible solution candidates for a service request (*matching*) and how to rate their relevance or importance (*ranking*).

Eventually, after the conceptual architecture has been established and fixed, one should do some evaluation and determine the major parameters and variation points which mainly affect and determine the performance of the systems and the extent to what the user expectations are met. All the available parameters should be documented in order to allow adaption and optimization of

[3]Even if this heuristics might be based on rather sophisticated techniques like natural language processing methods etc.

a concrete system that implements the architecture for a specific application domain.

The process itself roughly describes a transformation from the concrete inputs to the actual results that are computed when execution the process in a system. Hence, we need to define what information has to be fed into the process as input and what the process delivers in return as a result. In our case, we need to define what semantic- and syntactic-based information is used in the process and what information is included in a discovery request. A discovery request at least will contain some form of description what the user wants to achieve (the *goal* of the user) and perhaps some additional description on the *preferences* that a user has. Moreover, a methodology of how to specify and write down discovery requests and how to described services has to be defined. The results that are delivered by the discovery process are essentially determined abstract architecture and interpretable wrt. the formal meaning underlying the architecture and the single components.

The single components in the architecture are often not implemented from scratch but use different other external components to provide their intended functionality. In our case, the information on the single elements on which discovery is processed has to be stored in some form of registry or repository. Semantic components are likely to use some form of reasoning and thus some deduction system. Syntactic component could use external tools like index servers and natural language processing components.

A discovery process is always based on the some data about the elements that can be discovered as well as the specification of what should be retrieved. The discovery process can only be as good as these fundamental ingredients of the process and an actual system will only be widely accepted and used if it not only performs well by is comprehensible and useable (that means ergonomic). Thus, the development of adequate tools which support the different persons which are involved in the process is crucial: On the one hand, a service provider needs support to annotate its services and evaluate its descriptions. On the other hand, the clients who are looking for a service have to be supported in every possible way to conveniently express what they are looking for and to interpret and process the results of a discovery process. Clearly, the retrieval component is a essential core component of the retrieval system but all auxiliary tools are as important in order to have a successful system.

If one is really serious with building a system, then he has to do extensive evaluation of system characteristics and identify potential modifications of the system parameters such that these characteristics get enhanced. The most important aspects here are the quality of the result as well as the performance of the system. Clearly, one needs to find suitable measures for these aspects. A command standard approach for measuring the quality of results of retrieval systems for instance is *precision* and *recall*. Precision basically quantifies how many of the retrieved objects are actually relevant whereas recall measures how many of the relevant information are indeed being retrieved. During the last decades a variety of additional and domain-specific quality measures for retrieval systems have been developed. For computing these measures one has to do experiments or simulations. For these experiments one needs testbeds or even standardized benchmarks. The design and creation of such a testbed or benchmark is a non-trivial task which requires significant efforts.

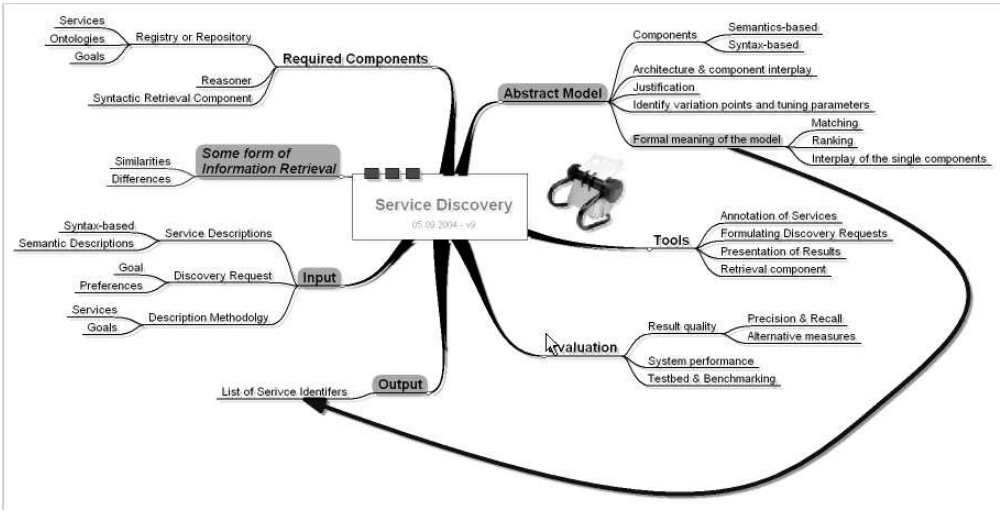

Figure 1: Relevant aspects of Service Discovery

In this document, we will mainly focus on the highlighted branches of the Mindmap in Figure 1, namely the abstract model of service discovery, how to annotate services and express goals, how to determine the result for a given discovery request and the discussion on the relationship between service discovery in WSMO and Information Retrieval.

1.3 Goal of this Document

Different applications and users of a system for the dynamic discovery of Web Services require and request different properties of the system and its underlying conceptual methodology: On the one hand, it should be easy to use and the data the system processes should be constructible with the least possible effort (for both, service providers and requesters), on the other hand, the system should provide reasonable performance (i.e. have high accuracy and low response times) to be actually useful for particular applications.

In this document, we aim at defining a comprehensive and flexible approach to service discovery in WSMO which integrates a variety of techniques with rather different semantic foundation, precision and computational complexity. Our final goal is to provide the foundation for a system which is adequate and beneficial for a wide range of applications and users:

WSMO and WSML provide the means to precisely describe web services functionality and requester goals using well-defined semantics, which is necessary to provide automation of web service discovery to any degree. However, authoring detailed and precise descriptions can be considerably complex and require a level of expertise in logics that might hamper their use in some real applications. In addition, the inferencing support required when using such precise definitions can have a high computational complexity; if a large number of web services is available for discovery, this can lead to unacceptable response times. Therefore, in this document we also explore other approaches that require less complex descriptions of services and requests and that can provide better response times when working with big repositories of web services.

The single techniques that are discussed in Section 2 are particularly useful for specific application scenarios and user. Considered as a comprehensive toolkit for Service Discovery, the can be used to construct complete an usable support for the discovery of web services in a wide range of scenarios. The choice of one of these techniques or their possible combination (for example, in order to reduce the number of services to be checked by the more complex but precise approaches) will be driven by the concrete application the discovery support is required for. Although the presented techniques operate at increasing levels of complexity wrt. to the requirements on logical reasoning a strict layering is not possible in general. Instead, the approaches are considered as being orthogonal to each other to some extent.

The presented approach covers the needs and requirements of a various possible application and usage scenarios. It is considered as the WSMO reference model for the dynamic location of WSMO/WSML-described web services that fulfill a requester goal.

1.4 Overview of the Document

In Section 2 we present and discuss the different approaches that can be useful for web service discovery in different scenarios. Finally, Section 3 draws some conclusions and describes our future work.

2 Principle Approaches to Web Service Discovery

In this section, we show and discuss the major approaches in a broad spectrum of techniques that can basically be used for discovering Web Services according to some description of a discovery request. The reviewed approaches mainly differ in their formal nature, the efforts for description of both, services and discovery requests, the accuracy of results that can be achieved by following them and the computation tractability and potential scalability.

2.1 Keyword-based Discovery

The keyword-based discovery is the first step in a complete framework for Semantic Web Service discovery. By performing a keyword base search the huge amount of available services can be filtered. Only the services retrieved in this first step will be "semantically checked" in the following steps of discovery. The focus of WSMO discovery is not in keyword-based discovery but we consider this kind of discovery a necessary step in a complete Semantic Web Service discovery framework.

In a typical keyword-based scenario a keyword-based query engine is used to discover services. A query, which is basically a set of keywords, is provided as input to the query engine. The query engine match the keywords from to query against the keywords used to describe the service. A query with the same meaning can be formulated by using a synonyms dictionary, like WordNet [4] [Fel98]. The semantic of the query remains the same but because of the different keywords used, synonyms of previous ones, more services that possible fulfill user request are found. Moreover, by using dictionaries like WordNet as well as natural language processing techniques an increase of the semantic relevance of search results (wrt. to the search request) can principally be achieved [RS95]; nonetheless, such techniques are inherently restricted by the ambiguities of natural language and the lack of semantic understanding of natural language descriptions by algorithmic systems.

As a result, a list of services identifiers (URLs) is returned. The services descriptions are stored in registries like UDDI registries. They are provided in the terms of advertisements along with the keywords for categorization.

2.2 Discovery with (Simple) Semantic Annotations

Although keyword-based search is a widely used technique for information retrieval, it does not use explicit, well-defined semantics. The keywords used to retrieve relevant information do not have an explicit formalization and, therefore, do not allow inferencing to improve the search results.

For these reasons, as a second approach we consider the use of controlled vocabularies with explicit, formal semantics. Ontologies, which offer a *formal, explicit specification of a shared conceptualization* [Gru93], can be used for this purpose. They provide an explicit and shared terminology to describe web services and requester goals, with an underlying logic formalization that enables the use of inference services.

First, we are interested in describing with ontologies the functional aspects of web services and requests and we look into existing proposals to do so. The annotation of non-functional properties will be discussed later in this section.

[4] The WordNet homepage: http://www.cogsci.princeton.edu/w̃n/

In OWL-S, the profile of a service can be positioned in a hierarchy of profiles[5]. In the following we show an example of such hierarchy[6]:

```
<owl:Class rdf:ID="E_Commerce">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-s/1.1B/Profile.owl#Profile"/>
  <rdfs:comment>Top level of an ontology of retail services;
    it requires a product to sell and a transportation mode</rdfs:comment>
  </owl:Class>

<owl:ObjectProperty rdf:ID="merchandise">
  <rdfs:domain rdf:resource="#E_Commerce" />
  <rdfs:range rdf:resource="#Product" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="deliveryMode">
  <rdfs:domain rdf:resource="#E_Commerce" />
  <rdfs:range rdf:resource="#Transportation" />
</owl:ObjectProperty>

<owl:Class rdf:ID="BookSelling">
  <rdfs:subClassOf rdf:resource="#E_Commerce" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#merchandise" />
      <owl:minCardinality
        rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#merchandise" />
      <owl:allValuesFrom rdf:resource="#Book" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="AirlineTicketing">
  <rdfs:subClassOf rdf:resource="#E_Commerce" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#merchandise" />
      <owl:allValuesFrom rdf:resource="#CommercialAirlineTravel" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

In this example, the general class of services selling and delivering products (e-commerce services) is specialized by the *BookSelling* class, which restricts the products sold to books, and the *AirlineTicketing* class, which restricts the products to commercial airline travels.

The classes of profiles defined in the above hierarchy, which characterize at a high level the functionality of services, can be used to position a given service in this hierarchy and to describe a user request. As the OWL definitions given have a formal semantics, subsumption reasoning can be applied to e.g. get also services that are more general than the kind of service requested.

In the METEOR-S project[7], WSDL descriptions are annotated with domain ontologies. WSDL operations are associated to a concept in an ontology of operations, and inputs, outputs, preconditions and effects of the operation are annotated with domain ontologies. The example below shows such annotations for a ticket booking service:

```
<wsdl:message name="OperationRequest">
  <wsdl:part name="in0" type="tns1:TravelDetails"
   LSDISExt:onto-concept="LSDISOnt:TicketInformation"/>
</wsdl:message>
<wsdl:message name="OperationResponse">
  <wsdl:part name="return" type="tns1:Confirmation"
```

[5] http://www.daml.org/services/owl-s/1.1B/ProfileHierarchy.html
[6] http://www.daml.org/services/owl-s/1.1B/ProfileHierarchy.owl
[7] http://lsdis.cs.uga.edu/Projects/METEOR-S/

```
   LSDISExt:onto-concept="LSDISOnt:ConfirmationMessage"/>
</wsdl:message>
<wsdl:portType name="Travel">
  <wsdl:operation name="buyTicket" parameterOrder="in0"
   LSDISExt:operation-concept="LSDISOnt:TicketBooking">
   <wsdl:input message="intf:OperationRequest" name="buyTicketRequest"/>
   <wsdl:output message="intf:OperationResponse" name="buyTicketResponse"/>
   <LSDISExt:precondition name="ValidCreditCard"
    LSDISExt:precondition-concept="LSDISOnt:ValidCreditCard"/>
   <LSDISExt:effect name="TicketBooked"
    LSDISExt:effect-concept="LSDISOnt:CardCharged-TicketBooked-ReadyForPickUp"/>
  </wsdl:operation>
  <wsdl:operation name="cancelTicket" parameterOrder="in0"
   LSDISExt:operation-concept="LSDISOnt:TicketCancellation">
   <wsdl:input message="intf:OperationRequest" name="cancelTicketRequest"/>
   <wsdl:output message="intf:OperationResponse" name="cancelTicketResponse"/>
   <LSDISExt:precondition name="CreditCardValidity"
    LSDISExt:precondition-concept="LSDISOnt:ValidCreditCard"/>
   <LSDISExt:precondition name="TicketBookedBefore"
    LSDISExt:precondition-concept="LSDISOnt:TicketExists"/>
   <LSDISExt:effect name="TicketCancelled"
    LSDISExt:effect-concept="LSDISOnt:CardCredited"/>
  </wsdl:operation>
</wsdl:portType>
```

The ontology of operations (possibly together with the annotation of the output) can serve as a simple characterization of the functionality of the service, where the explicit and formal semantics of the ontology concepts can be exploited to do subsumption reasoning. This is a very similar approach to the use of OWL-S profile hierarchies.

The annotation of inputs, outputs, preconditions and effects, both in METEOR-S and OWL-S, offers a characterization of these elements of the web service, and they can also be used during the discovery process. For example, [LH03] uses the annotation of the inputs and outputs of the service to match requests and services based on their subsumption (or common satisfiability) relation.

Here we see that the characterization of both the service functionality and their inputs and outputs can be useful for the location of relevant web services, by exploiting the formal and explicit semantics of the ontologies used for the annotation. This approach is relevant to WSMO discovery because it offers an interesting trade-off between the complexity of the descriptions and the precision of the search. If the appropriate actions and domain ontologies are in place, they can be used in an intuitive way by the user to define his request and by the provider to annotate his service. Furthermore, inference support can be exploited to automatically select services that e.g. exactly match the requester query, that are more general or more specific than the request, or that intersect it [LH03].

The discussed approach can be applied to WSMO web services in different ways:

- An ontology similar to the OWL-S profile hierarchy can be built, web service capabilities annotated with the concepts of this ontology, and goals can refer to these concepts to describe what kind of services the requester is looking for. This annotation can be done e.g. using the *dc:type* non-functional property from the WSMO core non-functional properties
- Such hierarchy can be defined by the WSMO goals, and *wgMediators* can position the service in the defined hierarchy.

We see an action/object model simple and intuitive to use for both requesters and providers. In this model, an ontology of actions is defined (buy, book, rent, etc), and the objects to which the actions are applied are taken from the available domain ontologies. For example, the *action* concept can be the root concept

of such an ontology of actions, with a property *object* with undefined range (or ranging over any concept). Concepts such as *buy*, *rent*, etc. would be defined as subconcepts of the *action* concept. The requester would define the kind of services he wants and the provider would describe the functionality the service fulfills by subclassing the concepts in the ontology of actions. For example, a requester (or a provider) can subclass the *buy* concept by restricting the range of the *object* property to the *book* concept, taken from an appropriate domain ontology.

As controlled vocabularies are used, this approach can be more accurate than keyword-based discovery and subsumption reasoning can be used to locate more general services than the kind of service specified in the request.

An open point here is how to incorporate this model into WSMO goals and capabilities. It can be done e.g. by subclassing of goals or by introducing new information in the goal definition describing the action and objects selected.

This issue will be discussed and decisions will be taken in a future version of this deliverable.

Regarding the use of inputs and outputs to discover matching services, we do not consider this option. The reasons are that this approach is less intuitive for requesters, that we believe that input information should not be provided at discovery time but only used when candidate services have been located, and that here more notions of match can be applied (see for example [LH03]), making the specification of the goal more complex for the requester.

When considering non-functional aspects of the service in the discovery process, the use of appropriate ontologies to annotate e.g. the coverage of the service can be again exploited to perform inferencing and get e.g. services which coverage is wider than the requested coverage.

The outlined approach provides an easy-to-use way of annotating services and requests, while benefiting from formal and explicit semantics and the support of existing inference engines. In particular, the use of hierarchies or ontologies of actions to annotate the services is very intuitive for the requester and the provider and can be exploited quite efficiently for reasoning.

However, as the web service functionality is not accurately defined, the results of the discovery process based on this approach are not guaranteed to actually fulfill the requester goal. In the next section, we add a new level of complexity which can overcome this limitation.

2.3 Discovery with Precise Descriptions

Using simple semantic annotations for a service as it it described in the previous section basically adds machine-processable semantic information to service descriptions which allows a discovery mechanism to exploit this semantics during the discovery process and deliver results with higher precision.

Nonetheless, the kind of semantic information that can be expressed in that approach is rather coarse-grained and limited wrt. the details that can be captured. Therefore, the precision that can be achieved by a discovery component is limited as well, if the user could already state more detailed information in the discovery request (and wants to do so). For some applications and users, in particular, if you aim at a high-degree of automation in a dynamic service-based software system, definitely an approach is needed which allows to model nuances

of meaning of the functionality of Web Services and the goal that a client wants to have resolved.

An elaborate description of the single main entities which are involved in the discovery process, namely service capabilities and goals, can be achieved by refining the pure conceptual level (where services and goals are basically concepts) as described in the Section 2.2. This requires at least a richer modelling language which allows to talk about objects in a world and hence variables, constant symbols and perhaps function symbols. That means, on this level be allow the use of a full First-order language (or possibly some restricted subset of such a language).

There are various ways of generalizing the previously discussed approaches:

(1) A minimalistic extension would be to move to the instance level and give a more detailed perspective on the extensions of the involved concepts. More precisely, we describe Services and Goals as concepts not by using concept expressions in some Description Logic but by *defining their extensions* by means of a *rich logical language* which includes variables (which goes beyond what common Description Logics can express). Semantic-based discovery then consists of establishing a certain well-defined semantic relationship (for instance a subset-relation of the concept extensions) between the concepts described by the service and the goals. The actual relationship to be used depends on the intention of the goal that a user has in mind.

(2) However, in the most elaborated form this means to consider a service as a *relation on an abstract state-space* and to capture the functionality provided by a service (as well as the goal to be resolved) in these terms. Here, we would consider for services input, output, preconditions, assumptions, postconditions and effects of a service, whereas in WSMO for goals we only consider the state of the world that is reached after the execution of a service and hence postconditions and effects.

In a sense, (1) is in its spirit closely related to the techniques outlined in Section 2.2 (namely, understanding and modeling the world in terms of sets of objects), whereas (2) is a conceptually different approach. Hence, we consider (1) as an intermediate step on a gradual shift from methods in 2.2 to methods which realize approach (2). It should not be hard to extend (1) such that it can cover (2) as well.

In the following, we will discuss approach (1) in more detail. Approach (2) will be discussed in the next version of the deliverable in detail as well.

Enriched DL-like Modelling. Description Logics are based on a set-theoretic semantics, that means expression in Description Logics represent sets of objects. Sets can be constructed from several other sets using specific operators. Formulae in Description Logics basically describe set-theoretic relations between such sets like set-inclusion.

Thus, in such a modelling approach the world or some problem domain is understood in terms of sets of objects and the main semantic properties that one is interested in is certain relationship between such sets and elements of the universe. From a knowledge representation perspective, this is a very natural and simple modelling approach for human beings[8]. Nonetheless, Description Logic

[8]Some experts even claim that this model coincides with the view on the world (and the things therein) that five year old children tend to have.

languages in general restrict the way one can characterize the set of instances of a concept (or its extension, resp.). This restriction is mainly based on the interest to have a guarantee of computational tractability: The relevant inferencing tasks in Description Logics, like subsumption or satisfiability checking, are decidable.

In the modelling approach that is outlined in this paragraph we want to exploit the same modelling style but give up the restriction of the concept language – we basically allow a rich language for describing sets of elements.

A web service can be seen as computational object which can be invoked by a client. The resulting execution of the web service generates (wrt. a set of input values) certain information as an output and achieves certain effects on the state of the world. Both, an output and as well as an effect can be considered as objects which can be embedded in some domain ontology. Goals can be seen as sets of elements which are relevant to the client as the outputs and the effects of a service execution. Thus and in line with the WSMO model, goals refer to the state of the world which is reached by executing some service.

According to this view, in the discussed approach services and goals are represented as sets of objects. The objects described in some service description and the objects used in some goal description might be interrelated in some way by an ontology or a set of ontology. Eventually, such an interrelation is needed to establish a matching between goals and services.

Clearly, the semantics of such set has to be clarified when it comes to matching service with goals, i.e. what does it mean that a certain objects are collected in a set in such descriptions or what has the user in mind when writing down such a set? As we will see shortly, the exact semantics is partly determined by the matching notion that is applied. For clients the selection of some matching notion itself represents a way to specify semantic details of their intention with a request. Thus, there will not be a single matching notion in this approach, but several different ones.

A service description captures the objects that can in principle be delivered by the service. Obviously, one has to options here: Either one only considers the objects which can be delivered by a *single execution* of the web service or the objects which can be delivered by executing the web service *any number of times*. For the moment, we want to abstract from concrete executions of web services (and hence inputs) and thus only consider the latter case, that means services describe abstractly what they can deliver (independent of any input and number of executions by the user) at all.

For simplicity reasons, we will consider in the following only outputs of a service and do not treat effects explicitly. The separation of effects and outputs in WSMO is a conceptual one and effects can basically be dealt with in basically the very same way.

Formally, we describe the above mentioned sets of outputs and effects by means unary predicates in a First-order Language. All elements of the universe which satisfy the predicate (according to its definition) are considered to be elements of the set.

The set of outputs of a service is defined by an unary predicate $ws(x)$ which is defined by a formula of the form

$$\mathcal{W} : \forall x.(\psi(x) \leftrightarrow ws(x)) \tag{1}$$

where $\psi(x)$ is an arbitrary first-order formula with at most one free variable x. In general, we assume that ψ somehow refers to some set of ontologies $\mathcal{O}$ (which can be thought of as being represented as a set of first-order sentences or a logical theory, resp.).

On the other hand, a goal describes a set of objects a user is interested in (either as output of a service or as an effect on the state of the world) when consulting a service.

Clearly, this explanation does not fully fix the precise meaning of the set, i.e. what it means that a set of objects is of interest for the client. Two rather natural options arise: either the client wants to express that he wants to get *all* the objects by executing a web service or he wants to have only *some* of them. We will see soon, that the corresponding intention is fixed by the concrete notion of matching that is selected by the user for discovering services[9].

Thus, a set of objects which are somehow relevant for the requestor is defined by an unary predicate $g(x)$ which is defined by a formula of the form

$$\mathcal{G} : \forall x.(\phi(x) \leftrightarrow g(x)) \tag{2}$$

where $\phi(x)$ is an arbitrary first-order formula with at most one free variable x. Again, ϕ might refer to some set of ontologies $\mathcal{O}$.

Now, we can formalize several types of *matching notion* which capture different forms of matching. The notions that we will discuss in the following are not new but well-studied in the literature, for instance in [LH03] in the context of service matchmaking based on Description Logics or [ZW97] in the context of component matching based on signatures. Similar to [LH03], we distinguish basically four types of notions[10]:

- **Exact-Match.** Here the delivered items of the service $\mathcal{W}$ and the set of relevant objects for the requestor as specified in the goal $\mathcal{G}$ perfectly match, i.e. they coincide.

  Formally, that means that we have to prove the following:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(g(x) \leftrightarrow ws(x)) \tag{3}$$

  where $\mathcal{W}$ is the definition of the web service, $\mathcal{G}$ is the definition of the goal, and $\mathcal{O}$ is a set of ontologies to which both descriptions refer.

  In other words, the service is able to deliver all relevant objects. No irrelevant objects will be delivered by the service.

  In this case we write $\mathcal{W} \equiv_{\mathcal{O}} \mathcal{G}$ to indicate this particular kind of match.

[9] In a way, the situation here is similar to problem of assigning a truth-value to some first-order formulae with free-variables. There, the question is in which way the variable is supposed to vary on a semantic level. The interpretation can be fixed on a meta-level (when giving a truth-definition) but certainly does not fit all possible scenarios. In principle, free-variables should be avoided in formulae. Instead, users should specify their intention with a variable clearly by *explicitly* fixing the variable's quantifier themselves.

[10] Here, we don't consider a so-called *Disjoint-Match* as in [LH03] since it describes actually a non-match between a service and a goal.

- **Subsumption-Match.** Here the delivered items of the service $\mathcal{W}$ is a subset of relevant objects for the requestor as specified in the goal $\mathcal{G}$.

  Formally, that means that we have to prove the following:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(ws(x) \rightarrow g(x)) \tag{4}$$

  where $\mathcal{W}$ is the definition of the web service, $\mathcal{G}$ is the definition of the goal, and $\mathcal{O}$ is a set of ontologies to which both descriptions refer.

  In other words, the service is able to deliver only relevant objects, but not necessary all of them.

  In this case we write $\mathcal{W} \sqsubseteq_{\mathcal{O}} \mathcal{G}$ to indicate this particular kind of match.

- **Plugin-Match.** Here the delivered items of the service $\mathcal{W}$ is a superset of relevant objects for the requestor as specified in the goal $\mathcal{G}$.

  Formally, that means that we have to prove the following:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(g(x) \rightarrow ws(x)) \tag{5}$$

  where $\mathcal{W}$ is the definition of the web service, $\mathcal{G}$ is the definition of the goal, and $\mathcal{O}$ is a set of ontologies to which both descriptions refer.

  In other words, the service is able to deliver all relevant objects, but might deliver objects which are considered as irrelevant for the goal, too.

  In this case we write $\mathcal{W} \sqsupseteq_{\mathcal{O}} \mathcal{G}$ to indicate this particular kind of match.

- **Intersection-Match.** Here the delivered items of the service $\mathcal{W}$ has a non-empty intersection with the set of relevant objects for the requestor as specified in the goal $\mathcal{G}$.

  Formally, that means that we have to prove the following:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists x.(g(x) \wedge ws(x)) \tag{6}$$

  where $\mathcal{W}$ is the definition of the web service, $\mathcal{G}$ is the definition of the goal, and $\mathcal{O}$ is a set of ontologies to which both descriptions refer.

  In other words, the service is able to deliver some relevant objects, but might deliver objects which are considered as irrelevant for the goal, too. It is only guaranteed that at least one such relevant objects will be delivered; the service can additionally provide as much irrelevant information as possible.

  In this case we write $\mathcal{W} \sqcap_{\mathcal{O}} \mathcal{G}$ to indicate this particular kind of match.

In the following, we want to briefly interpret and discuss these four defined notions to get a full understanding of the overall approach on extended DL-style modelling. In particular, we discuss the meaning of these notions in the context of the two afore mentioned intentions that a requestor could have when writing a goal in this set-based framework, namely whether the intention is that *all* relevant elements (*Intention* $\forall$) or only *some* of them are needed (*Intention* $\exists$). Obviously, this has a significant impact on whether a formal matching notion adequately models our understanding of a match in the real-world and to what extend the satisfaction of one of these formal notions means an actual match in the real-world.

Exact-match $\mathcal{W} \equiv_{\mathcal{O}} \mathcal{G}$. An exact match between a service description $\mathcal{W}$ and a goal description $\mathcal{G}$ describes a situation where the service is able to deliver all relevant objects and no irrelevant objects will be delivered by the service. Here, we have identified a service which can fulfill the needs of the requestor regardless of his intention with the goal description: If all relevant objects have to be delivered the service is suitable and if only some are needed the service in fact provides more than requested. Unfortunately, this situation is rather unlikely to happen given the fact that we basically aim at more detailed and fine-grained descriptions of services and goals. Obviously, $\mathcal{W} \equiv_{\mathcal{O}} \mathcal{G}$ holds if and only if $\mathcal{W} \sqsubseteq_{\mathcal{O}} \mathcal{G}$ and $\mathcal{W} \sqsupseteq_{\mathcal{O}} \mathcal{G}$.

Subsumption-Match $\mathcal{W} \sqsubseteq_{\mathcal{O}} \mathcal{G}$. Here the delivered items of the service $\mathcal{W}$ is a subset of relevant objects for the requestor as specified in the goal $\mathcal{G}$. That means it is not guaranteed that the service is able to delivers all relevant objects. Thus, under *Intention* $\exists$ for the goal description the service matches perfectly, whereas for *Intention* $\forall$, the service does actually *not* match; some elements might not be deliverable by the service. Nonetheless, the service can still be considered as potentially rather useful for the requestor since it delivers at least *some* of the requested information in this case (but no irrelevant information for the client at all). Hence, we consider an subsumption match under *Intention* $\forall$ as a partial match.

Clearly, a problem arises if the service description $\mathcal{W}$ is inconsistent, that means describes the empty set. Then, the service trivially meets the criterion (and thus potentially would be recognized as satisfying each goal) but actually does not deliver something. Thus, it should *not* be considered as matching.

In order to fix the problem, we adapt the definition 4 and include a consistency check as follows:

$$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(ws(x) \rightarrow g(x)) \wedge \exists x.(ws(x)) \tag{7}$$

Plugin-Match $\mathcal{W} \sqsupseteq_{\mathcal{O}} \mathcal{G}$. Here the delivered items of the service $\mathcal{W}$ is a superset of relevant objects for the requestor as specified in the goal $\mathcal{G}$. Again, under *Intention* $\exists$ for the goal description the service matches perfectly. Additionally, this notion formalizes a match under *Intention* $\forall$ as well, since all requested objects can be delivered by the service; but the service might deliver additional (irrelevant) information wrt. the given goal.

Once again, a problem arises if the goal description $\mathcal{G}$ is inconsistent, that means describes the empty set. Then, all service trivially meet the criterion (and thus potentially will be delivered as result of the discovery process) but actually the goal does not request something. Thus, the goal should *not* be considered as meaningful and no service should match.

In order to fix the problem, we adapt the definition 5 and include a consistency check as follows:

$$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(g(x) \rightarrow ws(x)) \wedge \exists x.(g(x)) \tag{8}$$

Intersection-Match $\mathcal{W} \sqcap_{\mathcal{O}} \mathcal{G}$. Here the delivered items of the service $\mathcal{W}$ is a superset of relevant objects for the requestor as specified in the goal $\mathcal{G}$.

What we have discussed right now, in fact can be considered as eight different formal criteria which can be used to formalize service discovery request according to a client's goal and his intention. More precisely, we have discussed the formal

notions

$$\mathcal{W} \equiv^{\forall}_{\mathcal{O}} \mathcal{G} \quad \mathcal{W} \sqsubseteq^{\forall}_{\mathcal{O}} \mathcal{G} \quad \mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G} \quad \mathcal{W} \sqcap^{\forall}_{\mathcal{O}} \mathcal{G}$$
$$\mathcal{W} \equiv^{\exists}_{\mathcal{O}} \mathcal{G} \quad \mathcal{W} \sqsubseteq^{\exists}_{\mathcal{O}} \mathcal{G} \quad \mathcal{W} \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G} \quad \mathcal{W} \sqcap^{\exists}_{\mathcal{O}} \mathcal{G}$$

where the superscript indicates the intention of the goal description and the subscript shows the dependency on some (possibly empty) set of ontologies $\mathcal{O}$.

As we have explained, some of these notions formalize our intuitive understanding of match adequately, whereas other notions only capture partial forms of matching between service and goal descriptions. The following table summarizes the discussion briefly:

Formal notion	Real-world		
	Match	Partial Match	No Match
$\mathcal{W} \equiv^{\forall}_{\mathcal{O}} \mathcal{G}$	yes	-	-
$\mathcal{W} \sqsubseteq^{\forall}_{\mathcal{O}} \mathcal{G}$	-	yes ($\mathcal{W}$ cons.)	yes ($\mathcal{W}$ incons.)
$\mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G}$	yes ($\mathcal{G}$ cons.)	-	yes ($\mathcal{G}$ incons.)
$\mathcal{W} \sqcap^{\forall}_{\mathcal{O}} \mathcal{G}$	-	yes	-
$\mathcal{W} \equiv^{\exists}_{\mathcal{O}} \mathcal{G}$	yes	-	-
$\mathcal{W} \sqsubseteq^{\exists}_{\mathcal{O}} \mathcal{G}$	yes ($\mathcal{W}$ cons.)	-	yes ($\mathcal{W}$ incons.)
$\mathcal{W} \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G}$	yes ($\mathcal{G}$ cons.)	-	yes ($\mathcal{G}$ incons.)
$\mathcal{W} \sqcap^{\exists}_{\mathcal{O}} \mathcal{G}$	yes	-	-

According to our discussion above, we can establish a partial order on this notions which reflects the accuracy of the notion wrt. the degree of coincidence with our real-world understanding of matching. We apply the following criteria for this order: Actual matches are preferred (smaller) over partial matches. Semantically stronger notions are preferred over weaker notions. Hence, the partial order $\preceq$ (where $a \preceq b$ means *a preferred over b*) on our notions[11] is a follows:

$$\begin{array}{cc} & \mathcal{W} \equiv^{\forall}_{\mathcal{O}} \mathcal{G}\,,\, \mathcal{W} \equiv^{\exists}_{\mathcal{O}} \mathcal{G} \\ \preceq & \mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G}\,,\, \mathcal{W} \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G} \\ \preceq & \mathcal{W} \sqsubseteq^{\exists}_{\mathcal{O}} \mathcal{G} \\ \preceq & \mathcal{W} \sqcap^{\exists}_{\mathcal{O}} \mathcal{G} \\ \preceq & \mathcal{W} \sqsubseteq^{\forall}_{\mathcal{O}} \mathcal{G} \\ \preceq & \mathcal{W} \sqcap^{\forall}_{\mathcal{O}} \mathcal{G} \end{array}$$

where the green criterions are considered as *matches* and the yellow criterions are considered as *partial matches* only. If non of the criterions apply then we consider the given goal and web service description as *non-matching*.

In fact, the green notions formalize situations that can basically be considered as matches between service descriptions and goals, whereby the single notions *reflect additional nuances of meaning* that a client can exploit to accurately model his desire:

The *Intersection-Match* reflects the minimum formal criterion for situations that can be considered as matching. Here, *no special additional information* for the search request (beyond what is given in the goal $\mathcal{G}$ and its corresponding

[11] Where we assume that the notion are adapted in such a way that they ensure that the inconsistency cases do not happen as we have explained before.

intention $I_{\mathcal{G}}$ of the goal). It can or should be used when it is only required to locate services which just fit.

The *Subsumes-Match* reflects additionally that *only relevant elements* will be delivered. Thus, if the user want insists on services which do not deliver irrelevant items which are not of interest for the client then he should decide to use a Subsumes-match for discovery.

The *Plugin-Match* on the other hand reflects additionally that *all relevant elements* can be delivered. But the matching service is not garantueed to not deliver additional. irrelevant information. In a sense, a service which matches a goal according to the Plugin-criterion can resolve the requestors desire in a complete manner. Thus, if the user want insists on services getting all the information as specified in the goal then he should decide to use a Plugin-match for discovery.

Finally, the *Exact-Match* combines the *Subsumes-* and the *Plugin-criterion* and thus garantuees that a service matching satisfying this criterion precisely fulfills what has been described in the goal; the service does neither deliver more nor less than requested. It has to be used by a client, when both completeness and correctness of the delivered objects are considered to be essential.

In conclusion, when formulating a discovery request, a user is supposed to describe the set of relevant objects, it's intention and the required (minimal) formal matching criterion. Every less-preferred criterion (according to our preference order $\preceq$) is not considered as to adequately capturing the user's desire. Every more-preferred criterion captures additional information and hence restricts the result set of the discovery process further.

The preference order $\preceq$ could additionally be used in cases where the selected criterion does not lead to matches in the discovery process to suggest some services which might potentially be useful for the requestor: If some user selects a notion n^I with intention I (where $I \in \{\forall, \exists\}$) for his discovery request and no matching services are found, then a discovery component could indicate this and instead provide all services which match the next $\preceq$-successor m^I, that means the next less preferred notion with the same intention.

Summary. As we have seen, the single notions can be used to describe different degrees of matching between service capabilities and the requested goal. Furthermore, they eventually represent the requestor's precise intention of the given goal modelling in his request. Hence, the matching notion to be applied in a specific discovery request should be selected by the user when creating the request.

Examples. Let us illustrate this approach with a few of examples from the travelling domain: we model some client goals and service capabilities related to flight tickets. We want to define the following informal sample goals and assume the requester wants to find a web service offering the following real world service:

$\mathcal{G}_1$: All flight tickets from any place in Ireland to any place in Austria.

$\mathcal{G}_2$: Flight tickets from any place in Ireland to any place in Ireland which is different form the starting location.

$\mathcal{G}_3$: All flight tickets from Galway to Dublin.

$\mathcal{G}_4$: Some flight tickets from Galway to any place in Ireland (one is enough, and the client does not necessarily care which one).

For the formalization of the example in our enriched set-based approach, we assume that an appropriate set of ontologies $\mathcal{O}$ for geographical data and purchasing are in place. In particular, we have a predicate $in(x, y)$ which states that object x is geographically located in object y and $flightticket(t, s, e)$ which states that t is a flightticket from location s to location e.

Hence, we can use the following definitions for the set of relevant objects:

$$\mathcal{G}_1 : \forall t.(g(t) \leftrightarrow \exists s, e.(in(s, ire) \wedge in(e, austria) \wedge flightticket(t, s, e)))$$
$$\mathcal{G}_2 : \forall t.(g(t) \leftrightarrow \exists s, e.(in(s, ire) \wedge in(e, ire) \wedge \neg(s \doteq e) \wedge flightticket(t, s, e)))$$
$$\mathcal{G}_3 : \forall t.(g(t) \leftrightarrow flightticket(t, galway, dublin))$$
$$\mathcal{G}_4 : \forall t.(g(t) \leftrightarrow \exists e.(flightticket(t, galway, e) \wedge in(e, ire)))$$

The corrsponding intentions of these set are as follows:

$$I_{\mathcal{G}_1} : \forall \qquad I_{\mathcal{G}_2} : \exists \qquad I_{\mathcal{G}_3} : \forall \qquad I_{\mathcal{G}_4} : \exists$$

Now let us further assume web services $\mathcal{W}_1, \ldots, \mathcal{W}_4$ available exposing the following informal capabilities:

$\mathcal{W}_1$ offers flights for any place in Europe to any place in Europe.

$\mathcal{W}_2$ offers flights from all places in Ireland to all other places in Ireland.

$\mathcal{W}_3$ offers flights from all places in Ireland to Innsbruck.

$\mathcal{W}_4$ offers flights from Galway to Dublin.

These capabilities are formalized as follows:

$$\mathcal{W}_1 : \forall t.(ws(t) \leftrightarrow \exists s, e.(in(s, europe) \wedge in(e, europe) \wedge flightticket(t, s, e)))$$
$$\mathcal{W}_2 : \forall t.(ws(t) \leftrightarrow \exists s, e.(in(s, ire) \wedge in(e, ire) \wedge \neg(s \doteq e) \wedge flightticket(t, s, e)))$$
$$\mathcal{W}_3 : \forall t.(ws(t) \leftrightarrow \exists s, e.(in(s, ire) \wedge flightticket(t, s, innsbruck)))$$
$$\mathcal{W}_4 : \forall t.(ws(t) \leftrightarrow flightticket(t, galway, dublin))$$

Now, given an appropriate set of ontologies $\mathcal{O}$ we want to check what goals match under which notions which services. We will first discuss one example in detail and summarize all remaining cases and their results in the next table.

Let us consider goal $\mathcal{G}_2$ and service $\mathcal{W}_1$:

$\mathcal{G}_2$ asks for flight tickets from any place in Ireland to any place in Ireland which is different form the starting location (and thus has an existential intention $I_{\mathcal{G}_2} = \exists$).

$\mathcal{W}_1$ offers flights for any place in Europe to any place in Europe.

They are formalized by

$$\begin{aligned} \mathcal{G}_2 &: \forall t.(g(t) \leftrightarrow \exists s, e.(in(s, ire) \wedge in(e, ire) \wedge \neg(s \doteq e) \wedge flightticket(t, s, e))) \\ \mathcal{W}_1 &: \forall t.(ws(t) \leftrightarrow \exists s, e.(in(s, europe) \wedge in(e, europe) \wedge flightticket(t, s, e))) \end{aligned} \tag{9}$$

If we now check for the Plugin-match with the same intention as $\mathcal{G}_2$, that means $\mathcal{W}_1 \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G}_2$, then we have to prove:

$$\mathcal{W}_1, \mathcal{G}_2, \mathcal{O} \models \forall x.(g(x) \rightarrow ws(x)) \wedge \exists x.(g(x)) \tag{10}$$

We check the first part of the conjunction in (10) as follows: Let o be an arbitrary object for which $g(o)$ holds, that means because of (9) it holds

$$in(s', ire) \wedge in(e', ire) \wedge \neg(s' \doteq e') \wedge flightticket(o, s', e')) \tag{11}$$

for some objects s' and e'.

We now check whether $ws(o)$ holds as well, that means:

$$\exists s, e.(in(s, europe) \wedge in(e, europe) \wedge flightticket(o, s, e)) \tag{12}$$

.

But this is obvious because of (11) for $s = s'$ and $e = e'$. Since o has been arbitrary, we have proven the first part of the conjunction.

Now, we check the second part of the conjunction, which states essentially

$$in(s', ire) \wedge in(e', ire) \wedge \neg(s' \doteq e') \wedge flightticket(o', s', e')) \tag{13}$$

for some objects s', e' and o'. This holds if we assume that the set $\mathcal{O}$ of used ontologies formalizes for instance the following statements:

- Ireland contains at least two distinct cities with airports
- Between each pair of cities with airports there is actually connection and thus a corresponding flight ticket.

These statements clearly hold (according to our understanding of the world) and thus we are done. That means we have proven $\mathcal{W}_1 \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G}_2$.

But our example and discussion clearly shows that our formal notions of matching are *inherently relative to some knowledge base* (some set $\mathcal{O}$ of ontologies). If the knowledge base does not contain the required information (i.e. is not adequate), then a formal matching can not be proven although intuitively (given our common understanding and knowledge about the real world) one would expect a match. As we have seen in our example as well, this problem occurs if we have to formally show the existence of some object with specific properties. This happens, for instance, in the consistency checking parts of the Plugin- and Subsumes-Matching notion or in the Intersection-Match criterion.

The following table summarizes the matching that one would intuitively expect. Obviously, they should be somehow captured in a suitable way in our framework:

Intuitive matching of $\mathcal{W}$ with $\mathcal{G}$	$\mathcal{W}_1$	$\mathcal{W}_2$	$\mathcal{W}_3$	$\mathcal{W}_4$
$\mathcal{G}_1$	match	no match	partial match	no match
$\mathcal{G}_2$	match	match	no match	match
$\mathcal{G}_3$	match	match	no match	match
$\mathcal{G}_4$	match	match	no match	match

Now, the following results are achieved with our formal matching notions defined above. Clearly, we only have to consider notions which capture the goal intention $I_{\mathcal{G}}$ of the user. In each case we give the $\preceq$-maximal matching notion which is satisfied (if it exists).

Formal matching of $\mathcal{W}$ with $\mathcal{G}$	$\mathcal{W}_1$	$\mathcal{W}_2$	$\mathcal{W}_3$	$\mathcal{W}_4$
$\mathcal{G}_1$	$\mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G}$	no criterion	$\mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G}$	no criterion
$\mathcal{G}_2$	$\mathcal{W} \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G}$	$\mathcal{W} \equiv^{\exists}_{\mathcal{O}} \mathcal{G}$	no criterion	$\mathcal{W} \sqsubseteq^{\exists}_{\mathcal{O}} \mathcal{G}$
$\mathcal{G}_3$	$\mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G}$	$\mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G}$	no criterion	$\mathcal{W} \sqsupseteq^{\forall}_{\mathcal{O}} \mathcal{G}$
$\mathcal{G}_4$	$\mathcal{W} \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G}$	$\mathcal{W} \sqsupseteq^{\exists}_{\mathcal{O}} \mathcal{G}$	no criterion	$\mathcal{W} \sqsubseteq^{\exists}_{\mathcal{O}} \mathcal{G}$

The example shows that our formal matching notions and their classification in *matching* and *partially matching* faithfully models our intuitive understanding of matching in the approach discussed here.

Intentions for Web Service Descriptions as well? The above described approach is in asymmetric in the sense that we essentially model goals as well as services in the same way (as simple sets of objects), but for only one of these items, we allow the author to give information about his intention, whereas for the other item, the intention is fixed: the sets of objects that describe a web service capability $\mathcal{W}$ is always interpreted universally (that means with Intention $I_{\mathcal{W}} = \forall$) as defining that *all* contained objects can be delivered by the service. Naturally, the question arises why this is so and why we do not extend this model such that it allows both, existential as well as universal intentions for web service capability descriptions as well.

We will argue that this is not needed since it is not really useful for practical cases. For this reason, we have not considered it in our discussion.

Despite, imagine a person advertising a web service by specifying a set of relevant objects $\mathcal{W}$ and declaring at the same time that the service can only deliver *some* of these objects.

What could we read off from such a description of a service when trying to determine a match between the service and a requesters goal? The answer obviously is: not too much, since we *do not know precisely what subset indeed can be delivered*! Hence, every goal with universal intention can never be matched and even in the case of an existential intention for the goal we can match anything since we do not know anything concrete about the supported subset.

One would have to express more details of the precisely supported subset to make something useful out of this approach. But this obviously corresponds to a web service description with universal intention where we exactly narrow the set of relevant objects to the set of actually delivered objects. Hence, considering existential intention for web service descriptions is completely useless.

Advantages and Disadvantages of the extended DL-style Model. We briefly summarize the main advantages and drawbacks of the explained approach:

+ This modelling approach is based on a very simple and intuitive perspective of the world where everything is considered in terms of sets (or concepts)

+ The approach allows in general description which are not possible in Description Logics and thus provides increased expressiveness. Common Description Logics like $\mathcal{SHIQ}(\mathbf{D})$ for instance can only express so-called *tree-like* first-order sentences [GHVD03] (a syntactically restricted form of the first-order language) whereas in the discussed approach arbitrary first-order formulae are allowed.

+ Because of the same conceptual modelling style, this approach potentially allows a seemless integration with descriptions based on standard Description Logics.

- This approach does not capture the actual relation between service input and the corresponding outputs. Thus, the semantics of a web service is only described in a conceptual manner and hence still rather coarse-grained (in fact to coarse-grained for enabling the automatation of the discovery and later execution of a service).

- The increased expressive power comes of course at the cost: Reasoning with these description is not necessarily guaranteed to be decidable anylonger. Nonetheless, this does not have to be a problem in applications. Furthermore, the proposed approach very flexible and allows to use syntactic restrictions[12] on the allowed definitions for web services and goals such that the main reasoning tasks are decidable too (e.g. formulae in the Guarded-Fragment).

2.4 Comparison

There are various approaches of how to capture the functionality of a services which have different properties wrt. efficiency of processing of these descriptions, their degree of formal foundation and the precision that can be achieved with such an approach. The simplest approach certainly is some form of annotation with keywords which potentially can lead to arbitrarily unprecise results but is very easy to handle, whereas using logical formulae which precisely capture the service functionality certainly is the most accurate way of describing a service but the most difficult and costly approach to deal with as well.

All the mentioned approaches have their benefits and disadvantages. They specifically support different applications and usage scenarios for dynamic Discovery of Services and are adequate for these scenarios, whereas for others they can not be considered to be actually useful.

Intermediate as well as combined techniques which are based on the afore mentioned approaches are imaginable and actually beneficial for a Service Discovery System which is supposed to serve a broad range of applications and users.

For this reason, the WSMO reference model for service discovery represents such an integrated approach. It will be discussed in more detail in section **??**.

[12] Different restrictions from the ones imposed by Description Logics and perhaps still more expressive.

3 Conclusions

In this version of the deliverable we basically have outlined various approaches on Discovery of Web Services with different requirements on the description of Web Services and the Discovery Request itself.

Clearly, this discussion and the comparison will be further elaborated in the next version of the document.

Our future work in this deliverable will focus on the overall design of the abstract framework for discovery in WSMO and the discussion of this framework, in particular, semantics-based approaches to Service Discovery.

4 Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto, and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme. The work is funded by the European Commission under the projects DIP, Knowledge Web, SEKT, SWWS and Esperonto; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme.

The editors would like to thank to all the members of the WSMO working group for their advice and input into this document.

References

[BCE+02] T. Bellwood, L. Clment, 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. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, July 2002.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley Pub Co, 2nd edition, 2003.

[BLFD99] T. Berners-Lee, M. Fischetti, and T. M. Dertouos. *Weaving the Web*. Harper, San Franscisco, 1999.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl, March 2001.

[DFvH03] J. Davies, D. Fensel, and F. van Harmelen, editors. *Towards the Semantic Web: Ontology-Driven Knowledge Management*. Wiley, 2003.

[Fel98] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, May 1998.

[FHLW03] Dieter Fensel, James Hendler, Henry Liebermann, and Wolfgang Wahlster, editors. *Spinning the Semantic Web*. MIT Press, 2003.

[GHVD03] B.N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary, 2003.

[Gru93] T. R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GS96] David Garlan and Mary Shaw. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[LH03] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology, 2003.

[RS95] Ray Richardson and Alan F. Smeaton. Using WordNet in a knowledge-based approach to information retrieval. Technical Report CA-0395, Dublin, Ireland, 1995.

[W3C03] W3C. Soap version 1.2 part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, June 2003.

[ZW97] Amy Moormann Zaremski and Jeannette M. Wing. Specification matching of software components. *ACM Trans. Softw. Eng. Methodol.*, 6(4):333–369, 1997.