WSML Deliverable

D5.1 v0.1

WSMO Discovery

WSML Working Draft – October 8, 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/v0.1/20041008`

Latest version:
`http://www.wsmo.org/2004/d5/d5.1/v0.1/`

Previous version:
`http://www.wsmo.org/2004/d5/d5.1/v0.1/20040906`

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.

Service discovery i.e. the automatic location of services that fulfill a user goal is a popular research topic. However, a proper conceptual setting for this task is mostly missing. Therefore, in this document we discuss certain conceptual issues in relation to service discovery. First, we identify three major steps in service discovery, strictly distinguish between service and web service discovery. Remarkably, only one of these steps is about web service discovery. Second, we discuss different techniques for web service discovery using this as a means for a better understanding of the dialectic relationship between discovery and mediation. Third, we will discuss in detail web service discovery based on the means of logic. In order to cover the complete range of scenarios that can appear in practical applications, several approaches to achieve the automation of web service discovery 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 discovery model presented in this paper constitute the necessary theoretical basis to achieve the automatic location of 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

The web is a tremendous success story. Starting as an in-house solution for exchanging scientific information it has become, in slightly more than a decade, a world wide used media for information dissemination and access. In many respects, it has become the major means for publishing and accessing information. Its scalability and the comfort and speed in disseminating information has no precedent. However, it is solely a web for humans. Computers cannot "understand" the provided information and in return do not provide any support in processing this information. Two complementary trends are about to transform the web, from being for humans only, into a web that interweaves computers to provide support for human interactions at a much higher level than is available with current web technology.

- The semantic web is about adding machine-processable semantics to data. The computer can "understand" the information and therefore process it on behalf of the human user (cf. [Fen03]).
- Web services try to employ the web as a global infrastructure for distributed computation, for integrating various applications, and for the automatization of business processes (cf. [ACKM03]). The web will not only be the place where human readable information is published but the place where global computing is realized.

The semantic web promises to make information understandable to a computer and web services promise to provide smooth and painless integration of disparate applications. Web services offer a new level of automatization in eWork and eCommerce, where fully open and flexible cooperation can be achieved, on-the-fly, with low programming costs. However, the current implementations of web service technology are still far from reaching these goals, as integrating heterogeneous and dynamically changing applications is still a tremendous task.

Eventually, semantic web services promise the combination of semantic web with web service technology in order to overcome the limitations of current web services by adding explicit semantics to them. The exploitation of such semantics can enable a fully mechanized web for computer interaction, which would become a new infrastructure on which humans organize their cooperations and business relationships (cf. [FB02]). OWL-S [The04] and WSMO [RLK04] are the major proposals for providing semantic annotations on top of a web service infrastructure.

An important step for fully open and flexible eCommerce would be the mechanization of service discovery. As long as human intervention is required in service discovery the potential costs of establishing a new eCommerce link may outrange the potential savings and advantages. Open, flexible, on-the-fly creation of new supply chains is essentially based on full or nearly full automatization of this process. Therefore, it is not surprising that automatic web service discovery is a popular research topic and many papers are published on it (cf. [AGDR03], [LH03a], [PKPS02a], [BHRT03], [GCTB01], [VSSP04], [SWKL02], [ZK04]). Still, many of these papers discuss discovery in the setting of multi-agent systems or in the setting of description logic based reasoning and none of them really seems to take a look on the actual conceptual and pragmatic issues that are involved in service discovery via the usage of web service technology.

Therefore, we provide an in-depth analysis of the major conceptual issues that are involved in service discovery via web services.

- First, we strictly distinguish between service and web service discovery, identifying three major steps in service discovery where only one of them is about web service discovery.

- Second, we discuss different techniques for web service discovery using this as a means for achieving a better understanding of the dialectic relationship between discovery and mediation. In this context, we discuss the mediation support needed for different approaches to web service discovery.

- Third, we discuss in detail logic based discovery of web services. Stepwise we will enrich the scenario we are able to support.

In conclusion, we provide a conceptual model for service discovery and different approaches to one of the steps of such model, web service discovery, which can be realized by WSMO and its related efforts.

The contents of this document are organized as follows. Section 2 identifies three major conceptual phases in service discovery. Section 3 analyzes the relationship between discovery and mediation based on different techniques to achieve web service discovery. Section 4 discusses in detail the issues around logic-based discovery of web services. Finally, Section 5 concludes the paper.

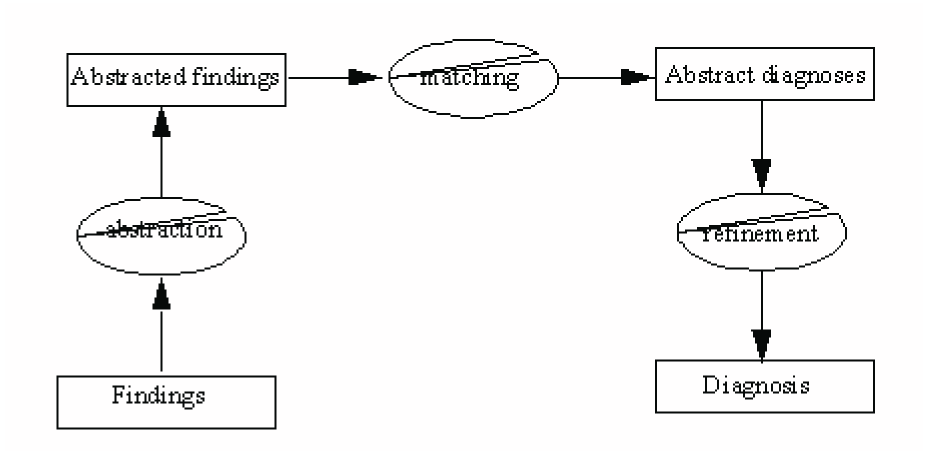

Figure 1: The three major processes of heuristic classification

2 A Conceptual Model for Service Discovery

In this section, we will first briefly introduce the gist of the matter of heuristic classification. This model is then applied to service discovery.

2.1 Heuristic Classification

[Cla85] provided a land marking analysis in the area of experts systems. Based on an analysis of xyz rule-based systems for classification and diagnosis he extracted a pattern of three inference steps that helped to understand the various production rules implemented in the various systems[1]. The problem-solving method he called *heuristic classification* separates abstraction, matching, and refinement as the three major activities in any classification and diagnosis task (see Figure 1).

Abstraction. Abstraction is the process of translating concrete description of a case into features that can be used for classifying the case. For example, the name of a patient can be ignored when making a diagnosis, his precise age may be translated into an age class, and his precise body temperature may be translated into the finding "low fever". The process is about extracting classification relevant features from a concrete case description.

Matching. Matching is the process of inferring potential explanation, diagnoses, or classifications from the extracted features. It matches the abstracted case description with abstract categories describing potential solutions.

Refinement. Refinement is the process of inferring a final diagnosis explaining the given findings. This process may include the acquisition of new features describing the given case. However, it is now the potential solution that guides the acquisition process of these features.

As the latest step indicates, the entire process can be executed in an iterative fashion. Instead of acquiring all potential findings an initial set can be used to

[1]In [OF01] we already used heuristic classification as a model for improving the translation process between different XML schema dialects

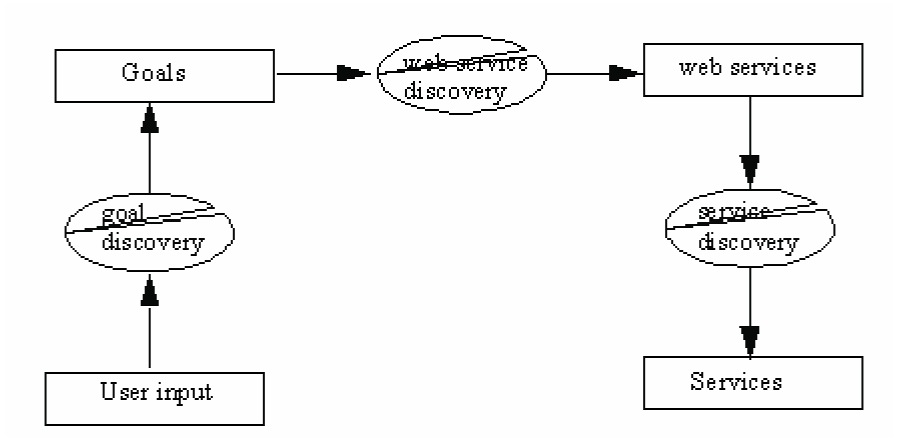

Figure 2: The three major processes in service discovery

derive intermediate potential explanation that can be further used to guide the next iteration of the process.

2.2 Service Discovery

Now, what has this to do with web service or service discovery? We strongly believe that a scalable and workable service discovery approach has to follow the same pattern (see Figure 2).

Abstracting goals from user input. Users may describe their goals in a very individual and specific way that makes immediate mapping with service descriptions very complicated. Therefore, each service discovery attempt requires a process where user expectations are mapped on more generic goal descriptions. Notice that this can be hidden by the fact that a discovery engine allows the user only to select from predefined goals. However, then it is simply the user who has to provide this mapping i.e. who has to translate his specific requirements and expectations into more generic goal descriptions. This step can be called **goal discovery**, i.e., the user or the discovery engine has to find a goal that describes (with different levels of accuracy) his requirements and desires. In the current literature on service and web service discovery this step is mostly neglected.

In order to fully understand the difference between matching and refinement in the service discovery context we have to introduce the difference between services and web services (cf. [Pre04]) and, in consequence, between service and web service discovery. Let us take a travelling scenario as a means to illustrate the difference. A customer may want to travel from A to B and he is looking for a service that provides this to him. The service may be provided by an airline or a train company. This is the service he is looking for. In order to find (and buy) the service he is accessing a web service, i.e., a software artifact. This software artifact will not provide him the service to travel from A to B (for this he needs a plane or a train) but it may help him to find this service. He will find a suitable web service based on the semantic annotations of available web services.

Actually, web services are means to find (and buy) services, i.e., they are to a large extent **service discovery engines and a discovery engine should not try to replace or duplicate this functionality**[2]. Using a second example, when somebody is looking for a book, he looks for a web service which is selling books based on the semantic annotations of the web service and he then consults the web service to check whether this book is in stock, with which price, and with which delivery dates and conditions. Taking the analogy with databases as illustration, web service discovery is about searching for databases that may contain instance data we are looking for, while service discovery is about finding the proper instance data by querying the discovered databases.

Unfortunately, most approaches around service and web service discovery neglect this distinction leading to non-pragmatically assumptions and non-workable proposals for the discovery process. Assuming to find services based on the semantic annotation of web services requires complete and correct meta descriptions of all the services offered by a web service e.g. a book-seller has to include in the web service description information about all the books in stock, their prices, etc. This would imply that the semantic annotation and the discovery process duplicate most of the actual service of the web service. It is no longer necessary to execute a web service to find a service, rather the semantic annotation and the related reasoning provide this support.

We do not think that complete and correct descriptions of all the services offered by a web service are a realistic assumption; it would make web services no longer necessary (at least for the discovery process); and we wonder whether logical reasoning would scale under this conditions where the execution of an efficient program accessed via the web service is simulated by reasoning over its semantic annotation.

Alternatively one could assume to directly query the web service during the web service discovery process. However, this may lead to network and server overload and it makes a very strong assumption: all mediation and wrapping service for the web service must be in place before the discovery process even starts. We do not think that this is a realistic assumption as we will further discuss in Section 3. In consequence we think it is essential to distinguish between web service and service discovery for coming up with a workable approach that scales and makes realistic assumptions in a pragmatic environment.

Web service discovery. Web service discovery is based on matching abstracted goal descriptions with semantic annotations of web services. This discovery process can happen on an ontological level, i.e., it can rely on conceptual and reusable terminologies for two reasons. First, the concrete user input has been generalized to more abstract goal descriptions. Second, concrete services and their descriptions have been abstracted to the classes of services a web service can provide. We believe that this twofold abstraction is essential for lifting web service discovery on an ontological level that is the prerequisite for a scalable and workable solution for it. In Section 4, different approaches to web service discovery will be discussed in detail.

Service discovery. Service discovery is based on the usage of web services for discovering actual services. Web service technology provides automated interfaces to the information provided by software artifacts that is needed to find, select, and eventually buy a real-world service or simply find the piece of information somebody is looking for. Service discovery requires strong mediation and wrapping, since the specific needs of a choreography of a web service have

[2]Spoken more precisely, web services are the interface to a software artifact that may help to find and buy services (cf. [Pre04]), however, we neglect this distinction here and try the web service interface and the accessed software artifact identically.

to be met in order to interoperate with it. Notice that automatization of service discovery defines significant higher requirements on mediation than web service discovery, as it also requires protocol and process mediation. In a sense, the role of web service discovery can be compared with the role of an internet search engine like Google, and service discovery with the process of extracting the actual information from the retrieved web sites.

3 Relation between Discovery and Mediation

In a distributed environment, different users and web services can use different terminologies, which leads to the need for mediation in order to make heterogeneous parties communicate. In this section we analyze the relation between discovery and mediation, identifying what kind of mediation is required in different scenarios.

3.1 Assumptions on Mediation

One could assume that web services and goals are described by the same terminology. Then no mediation problem exists during the discovery process. However, it is unlikely that a potentially huge number of distributed and autonomous parties will agree before-hand in a common terminology.

Alternatively, one could assume that goals and web services are described by completely independent vocabularies. Although this case might happen in a real setting, discovery would be impossible to achieve. In consequence, only an intermediate approach can lead to a scenario where neither unrealistic assumptions nor complete failure of discovery has to occur. Such an scenario relies in three main assumptions:

- Goals and web services use most likely different vocabularies, or in other words, we do not restrict our approach to the case where both need to use the same vocabulary.
- Goals and web services use some controlled vocabulary or ontology to describe requested and provided services.
- There is some mediation service in place. Given the previous assumption, we can optimistically assume that a mapping has already been established between the used terminologies, not to facilitate our specific discovery problem but rather to support the general information exchange process between these terminologies.

Under these assumptions, we do not simply neglect the mapping problem by assuming that it does not exist and, at the same time, we do not simply declare discovery as a failure. We rather look for the minimal assumed mediation support that is a pre-requisite for successful discovery.

Notice that this has also been the approach taken in IBROW (cf. [BPM$^+$98]), a land marking project in the area of internet-based matchmaking of task descriptions and competence definitions of problem-solving methods. Both tasks and methods used different ontologies to describe their requests and services. However, both description ontologies were grounded in a common basic ontology that allowed theorem proving to rewrite the terms until equality could be proven.

3.2 Mediation requirements

In the following, we will discuss different technological scenarios for the discovery process and how this reshapes our assumption on the underlying mediation support that needs to be in place.

3.2.1 Natural Language Processing and Keywords

Processing written natural language input of a human user is commonly used to derive information for computer consumption. Stemming, part-of-speech tagging, phrase recognition, synonym detection etc. can be used to derive machine-processable semantics from service requests and service descriptions. In the end, a set of keywords extracted from a service request are matched against a set of keywords extracted from a service description, or both requester and provider can use keywords to describe their requests and offers.

Stemming and synonym recognition already try to reduce different words to their joined underlying conceptual meaning. Still, this mediation support is very generic and only relies on general rules of language processing. Mediation can also be required for translating from a given human language into another one. Also in this case the mediation support required is generic and it can be assumed that it will be in place.

3.2.2 Controlled vocabularies

A different scenario is to assume that requester and provider use (not necessarily the same) controlled vocabularies[3]. In order to illustrate such scenario we will refer to controlled vocabularies for products and services. Efforts like UNSPSC[4] or eCl@ss[5] (among others) provide controlled vocabularies for describing products and services with around 15,000 concepts each. A service is described by a reference to a class in one of the classification schemas and a web service is described by the set of the classes that are used to describe the services that can be accessed through it. Similarly, a goal is described by a concept taken from a controlled vocabulary.

A goal discovers a web service if its concept is an element of the set of concepts describing the web service. Reasoning over hierarchical relationships may be required in case of taxonomies.

Mediation service is needed in case the requester and provider use different vocabularies. Since they use controlled instead of ad-hoc vocabularies, requiring such mediation service to be in place is not a completely unrealistic assumption. Notice that we do not assume a customized mediation service for our specific discovery task but rather generic alignments of generic business terminologies that may fulfill general information exchange needs between services and processes using these terminologies[6].

3.2.3 Ontologies

The border between controlled vocabularies and ontologies is thin and open for a smooth and incremental evolvement. Ontologies are consensual and formal conceptualizations of a domain (cf. [Gru93]). Controlled vocabularies organized in taxonomies and with defined attributes like eCl@ss resemble all necessary elements of an ontology. Ontologies may simply add some logical axioms for further restricting the semantics of the terminological elements. Notice that a service requester or provider gets these logical definition "for free". He can select a couple of concepts for annotating his service/request, but he does not need to

[3]Notice that this does not necessarily imply that service provider and requester use directly these controlled vocabularies. A goal discovery process and an analog process of discovering generic service descriptions can provide this service. Trials to mechanize such a process are described in [DKO+02].

[4]http://www.unspsc.org/

[5]http://www.eclass.de/

[6]Establishing this alignments is not an easy and straight-forward task as discussed in [dBP04].

write logical expression as long as he only reuses the ones already included in the ontology.

This scenario looks quite appealing since it adds semantic web facilities to the web service discovery preventing two major risk factors of logic-based techniques:

- Effort in writing logical expressions. Writing down correct logical formulas is a very cumbersome process. A discovery approach that is assuming this on a large scale for requesters and provides leads to scalability problems.
- Effort in reasoning about logical expressions. Reasoning over complex logical statements is computationally very hard in case the formal language provides a certain level of expressivity. Therefore, it is important to decouple the reasoning process from the actual discovery process. With the discussed approach this is possible, as the goal as well as the service descriptions are abstracted from concrete specifications and inputs to a generic description at the level of an ontology. The logical relationship between these concepts can be derived independent from a concrete service request and the materialized inferences can simply be reused during a discovery process. Doing reasoning in the back-end mostly or totally decoupled from the actual discovery request ensures that reasoning does not become a bottleneck.

Similar to Section 3.2.2, mediation support is needed in case the requester and provider use different ontologies. However, since generic ontologies can be used, such an existing mediation service is not a completely unrealistic assumption. Again, we do not assume customized meditation service for our specific discovery task but rather generic alignments of generic ontologies[7].

3.2.4 Full-fledged logic

Simply reusing existing concept definitions as described in the previous section has the advantage of the simplicity in annotating services and in reasoning about them. However, this approach has limited flexibility, expressivity, and grain-size in describing services and request. Therefore, it is only suitable for scenarios where a more precise description of requests and services is not required. Furthermore, describing the functionality of a Web Service requires an expressivity which goes beyond the capabilities of current ontology languages. For these reasons, a full-fledged logic is required when a higher precision in the results of the discovery process is required.

Mediation can only be provided if the terminology used in the logical expressions is grounded in ontologies. Otherwise, it is impossible to proof the given logical relation between requests and service descriptions and discovery breaks down. Therefore, the mediation support required is the same as for ontology-based discovery (see Section 3.2.3).

[7]See [dBP04] for a survey on Ontology mapping and alignment.

4 Web Service Discovery

In this chapter we discuss different approaches to web service discovery in the WSMO framework which require different effort in annotation and description of both goals and services and deliver discovery results of different accuracy. Our focus in this document is primarily on *semantic-based approaches* to web service discovery. We believe that the different techniques altogether help to create a workable solution to the problem of web service discovery which addresses practical requirements and is based on realistic assumptions; our final goal here is thus ensure that the WSMO framework and its discovery component is adequate for a wide range of application scenarios with rather different requirements. As explained in Section 3, the single approaches can require different techniques for mediation. The most important technique for us certainly is ontology merging and alignment. The creation of mediators itself is outside the scope of this deliverable.

4.1 Keyword-based Discovery

The keyword-based discovery is a basic ingredient in a complete framework for semantic web service discovery. By performing a keyword base search the huge amount of available services can be filtered or ranked rather quickly. The focus of WSMO discovery is not in keyword-based discovery but we consider this kind of discovery a useful technique 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 [8] [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.

The services descriptions are provided in the terms of advertisements along with the keywords for categorization.

4.2 Discovery based on simple semantic Descriptions of Services

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

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

services and requester goals, with an underlying logic formalization that enables the use of inference services.

4.2.1 A set-based Modelling Approach

A widely-used and quite natural way for human beings to represent knowledge about some domain is object-oriented or frame-based modelling. One main characteristic of these approaches is that the problem domain (or the *universe*) is understood as a set of objects and single objects can be grouped together into sets (or *classes*). Each class captures common (syntactic and semantic) features of their elements. Features can be inherited between classes by defining class hierarchies. This way, a problem domain can be structured as sets of objects and is basically understood as a collection of sets of things. In particular, Ontologies are a popular knowledge-representation technique which follow the very same modelling paradigm.

In such modelling approaches the main semantic properties that one is interested in are certain relationships between such sets or elements of the universe. Establishing and checking such relationships is the main reasoning task which allows agents to exploit knowledge formalized in the domain model (or Ontology). From a knowledge representation perspective, this is a very natural and simple modelling approach for human beings[9].

Description Logics [BCM$^+$03] are important examples for formal languages which are directly based on this idea and actually provide a language for describing such sets and relations between these sets: These languages 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 (set-theoretic) operators. Formulae in Description Logics basically describe set-theoretic relations between such sets like set-inclusion. Nonetheless, Description Logic languages in general restrict the way one can characterize the set of instances of a concept (i.e. its *extension*). 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 do not enforce any specific restriction on the concept language – we basically allow a rich (first-order) language for describing sets of elements. For concrete applications it is then up to the designers to choose an appropriate language (and a corresponding reasoning system) that fits their needs wrt. expressivity and computational characteristics. This could be for example a Description Logic like $\mathcal{SHIQ}$ or $\mathcal{SHIQ}(\mathbf{D})$ [HPSvH03], a full first-order Language like WSML-Full [de 04] or restricted sublanguages of thereof.

How to model Web Services and Goals? 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

[9]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.

line with the WSMO model [LRK04], goals refer to the state of the world which is desired by executing some web service.

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

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 two 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. We will come back on this issue in a later section where we discuss how to model the semantics of a service in a detailed and precise manner.

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. Nonetheless, it is useful to distinguish both since we believe that user will need the ability to apply *different* matching notions to outputs as well as effects.

Formally, we describe the above mentioned sets of outputs and effects by means of 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).

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 wrt. discovery, i.e. what it actually means in discovery that a set of objects „is of interest" for the client. Two quite 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[10].

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 [LH03b] in the context of service matchmaking based on Description Logics or [ZW97] in the context of component matching based on signatures. Similar to [LH03b], we distinguish four types of notions[11]:

- **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.

- **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.

[10]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, i.e. if is interpreted as a universal or as a existential placeholder. 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.

[11]Here, we don't consider a so-called *Disjoint-Match* as in [LH03b] since it describes actually a non-match between a service and a goal.

- **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 can 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 criterias to get a full understanding of the overall approach on the set-based modelling. In particular, we discuss the meaning of these criterions 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 (*Intention* $\exists$) are needed or requested by the client. 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.

Potentially, this leads to eight notions of matching which capture slightly different nuances of meaning of a discovery request by the client. Eventually, it is up to the client has to decide which notion of matching models his real-world desire adequately.

Exact-match $\mathcal{W} \equiv_{\mathcal{O}} \mathcal{G}$. 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. 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 $\mathcal{W} \sqsubseteq_{\mathcal{O}} \mathcal{G}$. 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 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 a subsumption match under *Intention* $\forall$ as a *partial match* since it contributes to the client's goal but does not fully satisfy it. The client has to use additional services to resolve his desire completely.

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 anything. 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 $\mathcal{W} \sqsupseteq_{\mathcal{O}} \mathcal{G}$. Again, under *Intention* $\exists$ for the goal description the service matches perfectly; we even get some *additional information* that the service is even able to deliver *all* relevant objects (not only some of them). 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 web services 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 $\mathcal{W} \sqcap_{\mathcal{O}} \mathcal{G}$. If a web service satisfies this criterion that means it can deliver some relevant object for the user and thus in the case of *Intention* $\exists$ we still have a perfect match. On the other hand, we don't have an actual match under *Intention* $\forall$ but only a very weak form of partial match, since the service can (very weakly) contribute to the user's desire.

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

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

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	-	-

Since we can adapt the formal criterions to handle the inconsistent cases in the right-most column properly, all the given notions formalize actual matches or partial matches in the real-world as we would expect. Hence, we can ignore the rightmost column.

As we explained above, in the case of *Intention* $\exists$ basically every of the given four criterions describes a match, whereas under *Intention* $\forall$ partial matches can happen too. Does it then actually make sense in the former case to distinguish the four existential notions at all? Basically, the most basic notion of matching under *Intention* $\exists$ is the intersection match $\sqcap^{\exists}_{\mathcal{O}}$. All other criteria ($\sqsubseteq^{\exists}_{\mathcal{O}}$, $\sqsupseteq^{\exists}_{\mathcal{O}}$, $\equiv^{\exists}_{\mathcal{O}}$) add only *additional information* on top of the intersection match which is only related to what additional properties a service would guarantee. Often, this might not be useful for the user directly, but we believe that this might be interesting for a discovery component when it comes to ranking services that match a service request using a notion under *Intention* $\exists$.

Please note, that the Intentions $(\exists, \forall)$ are not reflected in the formal criterions to be checked themselves but on a meta-level (in what we consider as a match or a partial match in the real-world).

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[12] is as follows:

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

where the green criterions are considered as *matches* and the yellow criterions are considered as *partial matches* only. If none of the criterions apply then we consider the given goal and web service description as *non-matching*.

[12]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.

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 (but does not have to) 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 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 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 irrelevant information additionally. 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 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 $\mathcal{G}$, its intention and (possibly) the required formal matching criterion with the same intention. 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.

In case that a client actually does not want to exploit additional meaning in his discovery request besides the goal $\mathcal{G}$ and its intention ($\exists$ or $\forall$) then he (or the discovery component) just selects (according to our preference order $\preceq$) the least-preferred criterion with the same intention that describes a full-match, i.e. $\sqcap^{\exists}_{\mathcal{O}}$ or $\sqsubseteq^{\forall}_{\mathcal{O}}$

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.

Intentions for Web Service Descriptions as well? The above described approach is 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 an existential intention for web service descriptions is not needed, since it does not really seem to be useful for practical cases. For this reason, we have not considered it in our discussion: 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 (i.e. using an existential intention). 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 cannot match anything since we do not know anything concrete about the supported subset; hence, we can even not establish an intersection match, since we do not know anything concrete about the object that can be delivered by the service.

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 narrow the set of relevant objects to the set of actually delivered objects.

Moreover, a service provider is interested in ensuring that his web service actually can be found by interested users; he will more likely claim to deliver to much than to few; in particular, a requester will never be interested to give a description which can not be matched during discovery. Hence, considering existential intention for web service descriptions is not useful at all.

Advantages and Disadvantages of the set-based Modelling Approach. We briefly summarize the main advantages and drawbacks of the discussed modelling 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 represents a general framework which does not fix the language to be used for describing goals and web services. In particular, it allows in general description which are not possible to express using Description Logics and thus provides increased expressiveness[13] over purely Description Logic-based approaches like [PKPS02b, LH03a].

+ Because of the same conceptual modelling style, this approach potentially allows a seamless integration of descriptions formalized in different languages.

- This approach does not capture the actual relation between service input and the corresponding outputs. Thus, the semantics of a web service is

[13]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

only described in a conceptual manner. In fact, this can be too coarse-grained for enabling the automation of the discovery and later execution of a service.

- The increased expressive power comes of course at a cost: Reasoning with these description is not necessarily guaranteed to be decidable any longer. Nonetheless, this does not have to be a problem in applications. Furthermore, the proposed approach is very flexible and allows to use restrictions[14] on the expressiveness of the allowed definitions for web services and goals such that the main reasoning tasks are decidable too (e.g. formulae in the Guarded-Fragment).

Indeed, the first mentioned disadvantage is not a real issue for people who are interest to model the service capability in detail: We will demonstrate later, that it is even possible to lift this modelling approach from the level of simple semantic annotations to the level of complex semantic descriptions where we describe a service capability by means of the actual relation between input values and generated outputs and effects.

4.3 Discovery based on rich semantic Descriptions of Services

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 high precision and recall.

Nonetheless, the kind of semantic information that can be expressed in that approach is limited wrt. the details of the service characteristics 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 certain 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 4.2. This requires at least a rich 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 we definitely want to use a First-order language (or possibly some restricted subset of such a language).

However, this is not enough. At this level of description we are interested to describe how outputs and effects created by a service execution actually depend on the concrete input provided by the user when invoking a service. That means to consider a service as a *relation on an abstract state-space* and to capture the functionality provided by a service in these terms. Here, we would consider for services input, output, preconditions, assumptions, postconditions and effects of a service, whereas in WSMO (version 1.0) for goals we only consider the

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

state of the world that is reached after the execution of a service and hence postconditions and effects.

In Section 4.3.1 we show how to extend the set-based modelling approach discussed in Section 4.2.1 to capture the actual relation implemented by the service as well. This will allow us to exploit the matching notions that we already discussed as well as give us additional notions that might be of interest in some situations.

In Section 4.3.2 we discuss a further approach which is based on Transaction Logic – an extension of First-order Logic that enables to specify the dynamics of logical theories (or knowledge bases) in a declarative way. Here, the state of the world is represented by a logical theory and since the execution of a service changes the state of the world[15] it results in an update of the logical theory. In principle, the described approach can be implemented using the $\mathcal{F}$LORA-2 system [KLP+04].

4.3.1 A set-based Modelling Approach for Rich Service Descriptions

In Section 4.2.1, we described the capability of a service by means of a set of objects and interpreted the set as a description of all objects that the service is able to deliver. In particular, we abstracted from concrete executions of the web service and the relation between the pre-state of the web service execution and the corresponding post-state. In other words, a web service description can be seen as a purely ontological concept whose description potentially can be quite detailed and complex.

In this section we show how to extend the set-based modelling approach discussed in Section 4.2.1 to capture the actual relation implemented by the service as well. That means, we skip the abstraction step that we have decided to take in Section 4.2.1 and consider service executions explicitly. Thus, we increase the level of detail of our service model.

The informal Service Model revisited. According to our discussion in Section 4.2.1 , a web service can be seen as computational object which can be invoked by a client. At invocation time, the client provides all the information needed by the web service to identify and deliver the concrete service that has been requested. 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.

So far we ignored inputs and their relations to outputs and effects of the web service execution and considered a web service as delivering a *single* set of objects. In fact, when considering actual executions of a web service, the sets describing the outputs and effects of the execution actually *depend on the provided input values.* Hence, a web service execution can be described by the sets of outputs and effects (for the *specific* input values), whereas a web service (which be seen as a collection of possible executions) should be modelled by a collection of set of outputs and effects: one for each service execution (or concrete input values).

Figure 3 illustrates the extended service model: When invoking a web service ws with some specific input values $i_1, \ldots, i_n$ in some state of the world (*pre-state*), the execution of the service results in a (different) state of the world (*post-state*) where the service delivers a set of elements as its output ($ws^{out}(i_1, \ldots, i_n)$) as well as a set of effects ($ws^{eff}(i_1, \ldots, i_n)$).

[15]At least of the information space!

In the WSMO framework the client specifies his desire as a goal. More precisely, the goal description consists of the specification of a set of desired information ($goal^{out}$) as well as a set of desired effects ($goal^{eff}$).

As before, matching mainly is based on checking certain set-theoretic relationships between the sets related to the output ($ws^{out}(i_1, \ldots, i_n)$ and $goal^{out}$) as well as the sets related to effects ($ws^{eff}(i_1, \ldots, i_n)$ and $goal^{eff}$). The interpretation of the various relationships between those sets has already been discussed in detail in Section 4.2.

Additionally, we can enrich our set of matching notions given in Section 4.2 by an orthogonal dimension[16]: We can express that we can satisfy a particular matching notion wrt. a *single execution* of a web service as well as wrt. an *arbitrary number of web service executions.* This results in additional matching notions that capture additional semantics in a given discovery request.

We want to illustrate the difference of the two new option by means of a simple example: Imagine the following goal a user

> ,,I want to know about all sports events in Innsbruck in the period of Sept 29th to Oct 14th 2004"

and a web service with the following capability

> ,,The service delivers all events for a given day in the current year in a given region in Austria"

Given appropriate domain knowledge about Austria, events, and dates, the service can not deliver the requested information by a single execution of the web service, but instead it can actually deliver *all* requested information, if it is invoked several times (i.e. for each day in the mentioned period of time and the region ,,Tyrol") if the current year corresponds to ,,2004".

As the examples shows, there might be situations, where such relaxed notions of matching actually can be useful for a user. Thus, we will discuss them here as well.

In the following paragraphs we will show how to formalize the extended web service model and discuss how to extend the matching notions from Section 4.2 to cover service executions. For the sake of simplicity, we will only mention one set of objects in the descriptions of goals and services. Handling the set of outputs and effects separately can be achieved in the very same way.

Formalizing the extended Service Model. Instead of using an unary predicate $ws(x)$ for describing the capability of a web service $\mathcal{W}$, we have to express the dependency of the objects delivered by a service on the concrete service execution and thus concrete input $i_1, \ldots, i_n$.

Let $\mathcal{W}$ be a web service with input parameters $i_1, \ldots, i_n$, then we formalize the capability of the service as follows:

$$\mathcal{W} : \forall x, i_1 \ldots i_n.(ws(x, i_1 \ldots i_n) \leftrightarrow \psi^{pre}(i_1 \ldots i_n) \wedge \psi^{post}(i_1 \ldots i_n, x)) \qquad (9)$$

where $\psi^{pre}(i_1 \ldots i_n)$ is an arbitrary first-order formula describing the precondition of the web service and $\psi^{post}(i_1 \ldots i_n, x)$ is an arbitrary first-order formula describing the postcondition. In $\psi^{post}(i_1 \ldots i_n, x)$ the variable x refers to the output value(s) generated by the service execution. The defined predicate $ws(x, i_1 \ldots i_n)$ has a natural interpretation: The value x will be delivered by the web service $\mathcal{W}$ if it is invoked with the input values $i_1 \ldots i_n$.

[16]This dimension precisely corresponds to the feature that we added when refining service model from Section 4.2: we can now talk about web service executions instead of a single abstract web service concept.

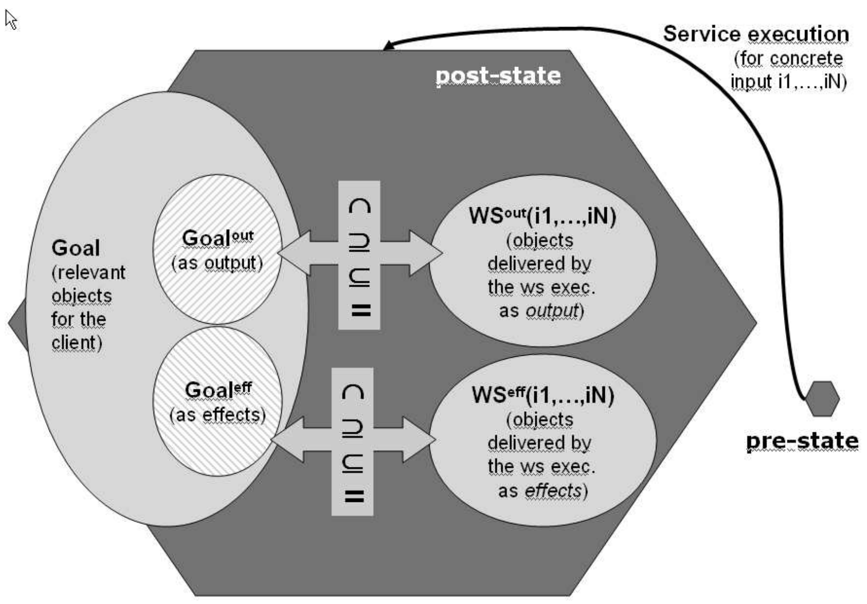
Service execution
(for concrete input i1,...,iN)
post-state
pre-state
Goal
(relevant objects for the client)
Goal^out
(as output)
Goal^eff
(as effects)
∩ ⊇ ⊆ =
WS^out(i1,...,iN)
(objects delivered by the ws exec. as output)
WS^eff(i1,...,iN)
(objects delivered by the ws exec. as effects)

Figure 3: An extended set-based Service Model.

Goals are just described in the very same way as in Section 4.2.1.

Adapting the Matching Notions. Since we adapted the way we describe web services, we have to adapt the formal criterions for our matching criterions as well. In the following we will show how to adapt the single notions accordingly and give a definition for the case in which we only consider *single executions* as well as the case of considering *multiple executions*.

This way, in principle we end up with (almost) 16 different notions of matching which potentially could be used by a client to specify his desire in a service request.

- **Exact-Match ($\mathcal{W} \equiv^{\forall}_{\mathcal{O}} \mathcal{G}$, $\mathcal{W} \equiv^{\exists}_{\mathcal{O}} \mathcal{G}$).**

  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 wrt. single execution of the web service $\mathcal{W}$:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n.(\forall x.(g(x) \leftrightarrow ws(x, i_1 \ldots i_n)) \qquad (10)$$

  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 this case we write $\mathcal{W} \equiv^{I;1-ex}_{\mathcal{O}} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

  Wrt. multiple executions we instead would have to prove the following:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(\exists i_1, \ldots, i_n.(g(x) \leftrightarrow ws(x, i_1 \ldots i_n)) \qquad (11)$$

  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 this case we write $\mathcal{W} \equiv^{I;n-ex}_{\mathcal{O}} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

- **Subsumption-Match ($\mathcal{W} \sqsubseteq_{\mathcal{O}}^{\forall} \mathcal{G}$, $\mathcal{W} \sqsubseteq_{\mathcal{O}}^{\exists} \mathcal{G}$).**

  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 wrt. single execution of the web service $\mathcal{W}$:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n.(\forall x.(g(x) \leftarrow ws(x, i_1 \ldots i_n)) \tag{12}$$

  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 this case we write $\mathcal{W} \sqsubseteq_{\mathcal{O}}^{I;1-ex} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

  Wrt. multiple executions we instead would have to prove the following:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(\exists i_1, \ldots, i_n.(g(x) \leftarrow ws(x, i_1 \ldots i_n)) \tag{13}$$

  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 this case we write $\mathcal{W} \sqsubseteq_{\mathcal{O}}^{I;n-ex} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

  Please note, that case of inconsistent definitions can be dealt with in the very same way as we discussed in Section 4.2.1.

- **Plugin-Match ($\mathcal{W} \sqsupseteq_{\mathcal{O}}^{\forall} \mathcal{G}$, $\mathcal{W} \sqsupseteq_{\mathcal{O}}^{\exists} \mathcal{G}$).**

  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 wrt. single execution of the web service $\mathcal{W}$:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n.(\forall x.(g(x) \rightarrow ws(x, i_1 \ldots i_n)) \tag{14}$$

  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 this case we write $\mathcal{W} \sqsupseteq_{\mathcal{O}}^{I;1-ex} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

  Wrt. multiple executions we instead would have to prove the following:

  $$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x.(\exists i_1, \ldots, i_n.(g(x) \rightarrow ws(x, i_1 \ldots i_n)) \tag{15}$$

  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 this case we write $\mathcal{W} \sqsupseteq_{\mathcal{O}}^{I;n-ex} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

  Again, the case of inconsistent definitions can be dealt with in the very same way as we discussed in Section 4.2.1.

- **Intersection-Match($\mathcal{W} \sqcap_{\mathcal{O}}^{\forall} \mathcal{G}$, $\mathcal{W} \sqcap_{\mathcal{O}}^{\exists} \mathcal{G}$).**

  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 wrt. single execution of the web service $\mathcal{W}$:

$$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n.(\exists x.(g(x) \wedge ws(x, i_1 \ldots i_n)) \qquad (16)$$

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 this case we write $\mathcal{W} \sqcap_{\mathcal{O}}^{I;1-ex} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

Wrt. multiple executions we instead would have to prove the following:

$$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists x.(\exists i_1, \ldots, i_n.(g(x) \wedge ws(x, i_1 \ldots i_n)) \qquad (17)$$

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 this case we write $\mathcal{W} \sqcap_{\mathcal{O}}^{I;n-ex} \mathcal{G}$ (with $I \in \{\forall, \exists\}$ the corresponding intention) to indicate this particular kind of match.

Obviously, the criterions (16) and (17) are logically equivalent and thus are the very same criterions. Thus, we do not have to distinguish between the two cases.

To conclude the discussion of the multiple notions we want to mention the following: The multiple execution notions actually check a set-theoretic relationship between the goal and the *union* of the sets of delivered objects over all possible (valid) inputs. Indeed, this can be considered as some sort of *abstraction from concrete executions of a web service* when checking a match and thus is very close to what we have discussed in Section 4.2.1. The main difference is there we do not consider explicit input parameters of a web service and do not refer to valid inputs only (i.e. refer explicitly to a precondition). Hence, we can consider the matching notions of this type as special cases of the ones that we discussed in Section 4.2.1.

4.3.2 Discovery with Transaction Logic

As discussed in previous sections, a more fine-grained matching requires a more precise description of what services a given web service offers access to, specifically of how the information and real-world effects described by the web service relate to the input provided to it.

In the following, we will denote goal postconditions and effects as G_{post} and G_{eff}, respectively. Similarly, we will denote the assumptions, preconditions, postconditions and effects of the capability of a given web service $Serv$ by $C_{ass}(Serv)$, $C_{pre}(Serv, Input)$, $C_{post}(Serv, Input)$, and $C_{eff}(Serv, Input)$. The preconditions, postconditions, and effects predicates include a parameter $Input$, meaning that the preconditions have to hold for the available input, and that the postconditions and effects will depend on the input given to the web service.

Information available to web service discovery We identify the following information available to the web service discovery process:

- Web services and goals are described in terms of a set of ontologies O, providing formal terminology which, as explained in Section 3, is amenable to automatic mediation. Such ontologies also provide the domain knowledge required to reason about the goal and web service descriptions.

- Additionally to the domain knowledge available to the web service discovery process, the requester may provide some specific knowledge which he is willing to use as an input to a Web Service to achieve his goal e.g. his particular credit card information. This kind of knowledge cannot be expected in the general case to be part of the set of ontologies O, which will only provide general domain knowledge e.g. that Innsbruck is located in Austria. However, this knowledge specific to the requester might be required by the web service to provide the advertised results and, furthermore, it is required in the matching process to check whether the web service has the necessary input information available to produce its results. We denote this requester specific knowledge by I.

- Some information might be also provided in the description of the goal itself e.g. if the goal requests a train ticket from Innsbruck to Frankfurt, this information can be used to prove our proof obligation for web service discovery. We denote such information by $In(G)$.

Proof obligations If the web service models the relation between its input and its results i.e. if the postconditions and effects delivered by the execution of the web service are dependent on the input information made available by the requester to the web service, we can check whether a given input information will lead to postconditions and effects that satisfy the postconditions and effects requested in the goal.

For this purpose, we first need to check that there exists input information $Input$ available to the web service i.e. contained in I or $In(G)$[17] that satisfies the preconditions ($C_{pre}(Serv, Input)$) of a potentially matching web service $Serv$.

If the above holds, we can hypothetically assume that the postconditions and effects of the web service for this input ($C_{post}(Serv, Input)$ and $C_{eff}(Serv, Input)$ hold. Under this assumption, we have the effects and postconditions of the candidate web service available; if the goal postconditions and effects hold in this state, it means that the web service provides the desired results for the available input and, therefore, $Serv$ is a match (for $Input$).

This can be formalized by the following proof obligation using transaction logic [BK98], where $\diamond$ is the hypothetical operator, and $\otimes$ is the sequence operator:

$$\begin{aligned} O, I, In(G) \models \exists Serv, \exists_{I,In(G)} Input \\ \diamond(C_{pre}(Serv, Input) \otimes insert\{C_{post}(Serv, Input), C_{eff}(Serv, Input)\} \\ \otimes\, G_{post} \wedge G_{eff}) \end{aligned} \tag{18}$$

In the formula above, $\exists_{I_In(G)} Input$ means, as explained before, that only information in I or $In(G)$ will be considered as possible input.

Since we use the hypothetical operator ($\diamond$), the assertion of the web service postconditions and effects will be rolled back i.e. retracted after the checking is finished.

The use of the sequence operator ($\otimes$) means that the preconditions must be tested before the postconditions and effects of the web service can be asserted, and that the goal will only be tested after this. In this way, we distinguish between the pre and post-state in the execution of the web service. This is especially relevant for checking the preconditions, as the assertion of the web

[17]Notice that the domain knowledge contained in O, although necessary for proving a match, is not necessarily input information explicitly made available by the requester to achieve his goal

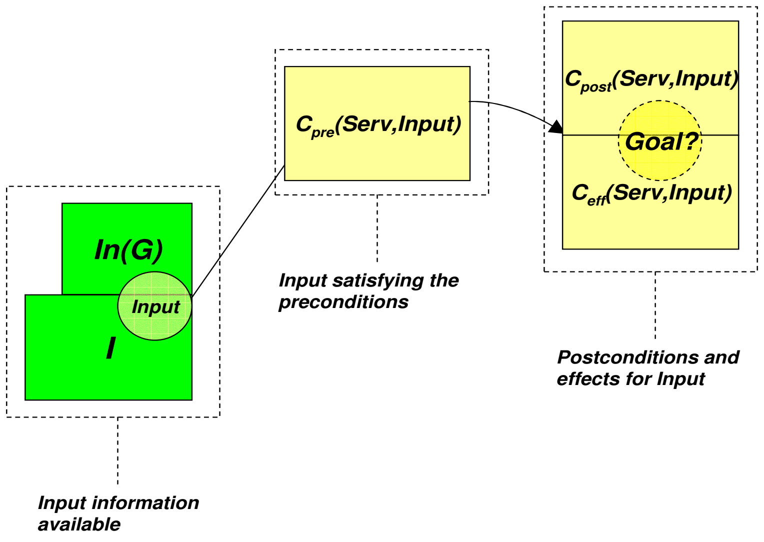
$C_{post}(Serv,Input)$
Goal?
$C_{eff}(Serv,Input)$
$C_{pre}(Serv,Input)$
In(G)
Input
I
Input satisfying the preconditions
Postconditions and effects for Input
Input information available

Figure 4: Hypothetically assuming web service postconditions and effects

service results will change the pre state[18] (both of the information and real-world spaces) and, by definition, the preconditions must be checked before such changes happen. Similarly, the goal must be tested only in the post-state i.e. when the postconditions and effects of the web service have been asserted.

The proof obligation is illustrated in Figure 4. On the left hand side of the picture, we represent the input information available. From this, $Input$ satisfying the web service capability preconditions $C_{pre}(Serv, Input)$ is considered. Depending on this input, we hypothetically assume (represented by an arrow in the figure) the postconditions and effects of the web service. These, represented by the box on the right hand side of the figure, are the results provided by the web service for the input considered. Finally, we check whether the goal is satisfied by these results i.e. if they contain the results requested in the goal.

Notion of match. For the description of the goal, we allow the use of both universally quantified goals i.e. all the results satisfying the conditions stated in the goal have to be delivered by the web service, as well as existentially quantified goals i.e. some results satisfying such conditions have to be delivered. Whether the request is existential or universal will be part of the description of the goal itself. If existential quantification is used, web services providing at least one result satisfying the conditions stated in the goal will be matched, while only web services providing all such results will be matched if universal quantification is used. In any case, the proof obligation (18) will provide a guarantee[19] that the selected services can fulfill the goal with the available input.

Single execution Vs multiple execution. The proof obligation (18) states that there exists and input ($Input$) available that fulfills the web service preconditions and that generate results that satisfy the goal. This means that the web service has to fulfill the goal with a unique tuple of inputs i.e. with a single execution.

[18]This is true for every web service which really provides a service. Otherwise, the web service does not have any interest for the requester and it should not even be considered.

[19]This guarantee is relative to the degree of accuracy in the description of the web service functionality. In the general case, as discussed in section 2, service discovery will be required for a complete guarantee.

If we want to allow a simple form of composition, matching services that can achieve the goal through the use of a set of inputs i.e. a set of invocations, we have to modify our previous proof obligation, which leads to proof obligation (19):

$$\begin{aligned} O, I, In(G) \models \exists Serv \\ \Diamond(insert\{C_{post}(Serv, Input), C_{eff}(Serv, Input)|_{I,In(G)} C_{pre}(Serv, Input)\} \\ \otimes\, G_{post} \wedge G_{eff}) \end{aligned} \tag{19}$$

$insert\{C_{post}(Serv, Input), C_{eff}(Serv, Input)|_{I,In(G)} C_{pre}(Serv, Input)\}$ corresponds in the previous formula to a *bulk update* i.e. the assertion of the postconditions and effects will occur for every Input in I or $In(G)$ satisfying the conditions in $C_{pre}(Serv, Input)$. Therefore, we hypothetically assume the results of the service for every possible input that satisfies the web service preconditions, and we test the goal with all such results hypothetically asserted i.e. we test the goal over the union of these results. This allows to match web services that require multiple executions to achieve the goal.

Considering domain knowledge as available input. We assumed above that the input available to the web service to provide its results is restricted to the information explicitly provided by the requester i.e. we restricted the input to I and $In(G)$. However, in some cases it might be of interest to also consider as possible input the instance data contained in the domain knowledge provided by the set of ontologies O e.g. that Innsbruck is an instance of *city* and it is *located in* Austria, which is an instance of *country*. In that case, there must exist $Input$ available in I, $In(G)$, or O that produces the requested results and, therefore, $\exists_{I_In(G)} Input$ must be replaced by $\exists Input$ in proof obligation 18, and $|_{I,In(G)}$ by $|$ in proof obligation 19.

As an example, consider as a goal getting the price for a museum pass for Innsbruck. Let us assume that there is a service offering museum pass prices for any city in Austria, and that the web service acting as an interface to the service requires as an input the country for what the information is requested and returns *museum pass prices for all the cities in Austria that offer a museum pass*. This web service, thus, provides the information wanted by the requester, but it requires as an input Austria, not Innsbruck. A match can be proved if we consider that information in O can be used as input to the web service i.e. if we consider that the Austria instance as a possible input to the web service.

Either if we include the information in O as available input or not, the requester will get as the result of the web service discovery process the set of matched web services together with the input necessary to achieve its request for each of such web services. As the results of a web service are dependent, in the general case, on the input provided to it, a complete match result must include not only matched web services but also **for what input the requested results will really be provided by the web service**. These inputs are the ones to be used in service discovery for executing the web service.

Assumptions. In proof obligations (18) and (19) we have not taken the web service assumptions into account. In this sense, different proof obligations can be defined for the case in which we also want to make sure that the assumptions of a web service $Serv$ hold to mach it, and for the case in which we do not want to check the assumptions. The latter case is formalized by the previous proof obligations. For the former, we have to introduce new knowledge that might be available to the web service discovery process, W, about the current state of the world not covered by neither I, $In(G)$, nor O[20]. The extended proof obligations for (18, 19) are (20, 21):

$$\begin{aligned} O, I, In(G), W \models \exists Serv\ C_{ass}(Serv) \wedge \exists_{I,In(G)} Input \\ \Diamond(C_{pre}(Serv, Input) \otimes insert\{C_{post}(Serv, Input), C_{eff}(Serv, Input)\} \\ \otimes\ G_{post} \wedge G_{eff}) \end{aligned} \tag{20}$$

$$\begin{aligned} O, I, In(G), W \models \exists Serv\ C_{ass}(Serv) \wedge \\ \Diamond(insert\{C_{post}(Serv, Input), C_{eff}(Serv, Input)|_{I,In(G)} C_{pre}(Serv, Input)\} \\ \otimes\ G_{post} \wedge G_{eff}) \end{aligned} \tag{21}$$

Relation to other approaches The approach presented in this section corresponds to an extension of set based modelling from Section 4.2.1 to include the relation between the web service results and its input. Roughly speaking, $C_{pre}(Serv, Input)$ and $C_{post}(Serv, Input)$ correspond to the parameterization of the sets of results provided by the web service with the input given.

Regarding the approach presented in section 4.3.1, the use of transaction logic introduces the main advantage of explicitly considering the state in the proof obligations for web service discovery. As discussed before, the preconditions of the web service must be checked in the pre-state i.e. before the postconditions and effects of the web service are generated. Not considering the state in the discovery process might lead to problems in some cases e.g. if the web service requires the balance of the requester's account to be higher than a given amount and one of its effects is to decrease the balance of such account. If preconditions and effects are checked in the same state, some expected matches might fail.

The notions of match identified in previous approaches are covered by the different proof obligations introduced in this section. Although partial matches i.e. $\mathcal{W} \sqsubseteq^{\forall}_{\mathcal{O}} \mathcal{G}$ and $\mathcal{W} \sqcap^{\forall}_{\mathcal{O}} \mathcal{G}$ are not directly covered, they can be relaxed to $\mathcal{W} \sqsubseteq^{\exists}_{\mathcal{O}} \mathcal{G}$ and $\mathcal{W} \sqcap^{\exists}_{\mathcal{O}} \mathcal{G}$, respectively. In fact, partially matching in the first two cases is equivalent to relaxing the intention of the requester from universal intention to existential intention. Therefore, we can see that this approach covers all the cases discussed in previous sections.

[20]Notice that in general the ontologies will only include stateless knowledge.

5 Conclusions

In this version of the deliverable we have defined a conceptual model for service discovery which avoids unrealistic assumptions and is suitable for a wide range of applications. One of the key features of this model is the explicit distinction between the notions of web services and services. Moreover, the model does not neglect one of the core problems one has to face in order to make discovery work in a real-world setting, namely the heterogeneity of descriptions of requestors and providers and the required mediation between heterogeneous representations. As discussed in Section 3, the discussed approaches are based on using terminologies, controlled vocabularies or rich descriptions which are based on ontologies. For each of them, a working solution to the mediation problem is possible or not unrealistic.

We have outlined various approaches on discovery of web services with different requirements on the description of web services and the discovery request itself. Our main focus has been on semantic-based approaches to web service discovery.

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

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

References

[ACKM03] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services.* Springer, 2003.

[AGDR03] Rama Akkiraju, Richard Goodwin, Prashant Doshi, and Sascha Roeder. A method for semantically enhancing the service discovery capabilities of UDDI. In Subbarao Kambhampati and Craig A. Knoblock, editors, *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 87–92, 2003.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2003.

[BHRT03] B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. Request rewriting-based web service discovery. In *The Semantic Web - ISWC 2003*, pages 242–257, October 2003.

[BK98] A.J. Bonner and M. Kifer. A logic for programming database transactions. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, March 1998.

[BPM$^+$98] V. R. Benjamins, E. Plaza, E. Motta, D. Fensel, R. Studer, B. Wielinga, G. Schreiber, Z. Zdrahal, and S. Decker. Ibrow3: An intelligent brokering service for knowledge-component reuse on the world-wide web. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW98)*, Banff, Canada, 1998.

[Cla85] William J. Clancey. Heuristic classification. *Artificial Intelligence*, 27(3):289–350, 1985.

[dBP04] Jos de Bruijn and Axel Pollers. Towards and ontology mapping specification language for the semantic web. Technical report, DERI, 2004.

[de 04] Jos de Bruijn (*editor*). The WSML Family of Representation Languages. Working draft, Digital Enterprise Research Insitute (DERI), March 2004.

[DKO$^+$02] Ying Ding, M. Korotkiy, Boris Omelayenko, V. Kartseva, V. Zykov, Michael Klein, Ellen Schulten, and Dieter Fensel. Goldenbullet: Automated classification of product data in e-commerce. In *Proceedings of Business Information Systems Conference (BIS 2002)*, Poznan, Poland, 2002.

[FB02] D. Fensel and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

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

[Fen03] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition*. Springer-Verlag, Berlin, 2003.

[GCTB01] J. Gonzlez-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. In *KI-2001 Workshop on Applications of Description Logics*, September 2001.

[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.

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 2003. To appear.

[KLP$^+$04] Michael Kifer, Ruben Lara, Axel Polleres, Chang Zhao, Uwe Keller, Holger Lausen, and Dieter Fensel. A logical framework for web service discovery. In *"Semantic Web Services: Preparing to Meet the World of Business Applications" worshop at ISWC 2004*, 2004.

[LH03a] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*, Budapest, Hungary, May 2003.

[LH03b] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology, 2003.

[LRK04] Holger Lausen, Dumitru Roman, and Uwe Keller (*editors*). Web service modeling ontology - standard (WSMO-Standard), version 1.0. Working draft, Digital Enterprise Research Institute (DERI), 2004. `http://www.wsmo.org/2004/d2/v1.0/`.

[OF01] Boris Omelayenko and Dieter Fensel. An analysis of integration problems of xml-based catalogs for b2b electronic commerce. In *Proceedings of the 9th IFIP 2.6 Working Conference on Database Semantics (DS-9)*, pages 232–246, April 2001.

[PKPS02a] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In I. Horrocks and J. Handler, editors, *1st Int. Semantic Web Conference (ISWC)*, pages 333–347. Springer Verlag, 2002.

[PKPS02b] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347. Springer-Verlag, 2002.

[Pre04] Chris Preist. A conceptual architecture for semantic web services. In *Proceedings of the International Semantic Web Conference 2004 (ISWC 2004)*, November 2004.

[RLK04] Dumitru Roman, Holger Lausen, and Uwe Keller (*editors*). Web Service Modeling Ontology (WSMO). Working draft, Digital Enterprise Research Insitute (DERI), September 2004. Available from `http://www.wsmo.org/2004/d2/v1.0/`.

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

[SWKL02] K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, pages 173–203, 2002.

[The04] The OWL Services Coalition. OWL-S 1.1 beta release. Available at http://www.daml.org/services/owl-s/1.1B/, July 2004.

[VSSP04] K. Verma, K. Sivashanmugam, A. Sheth, and A. Patil. Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *Journal of Information Technology and Management*, 2004.

[ZK04] O.K. Zein and Y. Kermarrec. An approach for describing/discovering services and for adapting them to the needs of users in distributed systems. 2004.

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