DIP
Data, Information and Process Integration with Semantic Web Services
FP6 – 507483

Deliverable

WP4: Service Usage
D4.8
D4.8 Discovery Specification

Andreas Friesen, Stephan Grimm

December 21, 2005
EXECUTIVE SUMMARY

The automated and dynamic discovery is a crucial building block within the Semantic Web Services Paradigm. Since the discovered web services have to be further processed (i.e., automatically selected, composed and executed) by an automated agent, the realization of automated discovery requires discovery techniques providing matches of a very high quality. Especially, any ambiguities concerning “what the discovered service does” must be avoided. We realize discovery through matching of semantic descriptions that describe functional properties of a web service. We show how matchmaking is realised by use of the reasoner delivered in WP1. In our formalization of service descriptions, we support two variants of WSML language, namely WSML Flight and WSML-DL. We also discuss the advantages and weaknesses of the introduced matchmaking approaches.

The semantic reasoning is very expensive in the sense of required computational resources and cannot be performed in a distributed way. This may cause serious performance problems if large sets of web services have to be analysed. Therefore, we introduce a distributed semantic discovery framework.

To link our technology to DIP use cases, we apply our discovery solution to concrete examples from the use case workpackages WP10 (e-banking) and WP8 (B2B Telecom).

This deliverable may be of potential interest to the following readers:

- WP1-requirements put on the reasoner technology in the discovery context
- WP2 - requirements for the repository interface for retrieving web service descriptions and ontologies as input parameters for the discovery process
- WP4 - usage of our component within the QoS-based discovery and composition component
- WP5 - upcoming issues on incorporating mediation into discovery, to be addressed in D5.7
- WP6 - integration of the discovery component into the WSMX architecture
- WP8, WP9, WP10 - usage of the discovery component in the context of a concrete use case scenario
- Other partners interested in semantic discovery

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners. The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
**Abstract (for dissemination)**

We realise discovery through matching of semantic descriptions that describe functional properties of a web service. We show how matchmaking is realised by use of the reasoner delivered in WP1. In our formalisation of service descriptions, we support two variants of WSML language, namely WSML Flight and WSML-DL. We also discuss the advantages and weaknesses of the introduced matchmaking approaches. The semantic reasoning is very expensive in the sense of required computational resources and cannot be performed in a distributed way. This may cause serious performance problems if large sets of web services have to be analysed. Therefore, we introduce a distributed semantic discovery framework. To link our technology to DIP use cases, we apply our discovery solution to concrete examples from the use case workpackages WP10 (e-banking) and WP8 (B2B Telecom).

**Keywords**

Semantic Web Services (SWS), Semantic Discovery, capability matching 

**Version Log**

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-12-2005</td>
<td>1</td>
<td>Andreas Friesen, Stephan Grimm</td>
<td>v 1.0</td>
</tr>
</tbody>
</table>

**Logo**

Deliverable 4.8  
iv  
December 21, 2005
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>National University of Galway</td>
<td>NUIG</td>
<td>Dr. Sigurd Harand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Digital Enterprise Research Institute (DERI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>National University of Ireland, Galway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Galway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ireland</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:sigurd.harand@deri.org">sigurd.harand@deri.org</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +353 91 495112</td>
</tr>
<tr>
<td>Fundacion De La Innovacion.Bankinter</td>
<td>Bankinter</td>
<td>Monica Martinez Montes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fundacion de la Innovation. BankInter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Paseo Castellana, 29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28046 Madrid, Spain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:mmtnez@bankinter.es">mmtnez@bankinter.es</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: 916234238</td>
</tr>
<tr>
<td>Berlecon Research GmbH</td>
<td>Berelcon</td>
<td>Dr. Thorsten Wichmann</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Berlecon Research GmbH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oranienburger Str. 32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10117 Berlin, Germany</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:tw@berlecon.de">tw@berlecon.de</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +49 30 2852960</td>
</tr>
<tr>
<td>British Telecommunications Plc.</td>
<td>BT</td>
<td>Dr. John Davies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BT Exact (Orion Floor 5 pp12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adastral Park Martlesham</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ipswich IP5 3RE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>United Kingdom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:john.nj.davies@bt.com">john.nj.davies@bt.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +44 1473 609583</td>
</tr>
<tr>
<td>Swiss Federal Institute of Technology,</td>
<td>EPFL</td>
<td>Prof. Karl Aberer</td>
</tr>
<tr>
<td>Lausanne</td>
<td></td>
<td>Distributed Information Systems Laboratory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>École Polytechnique Fédérale de Lausanne</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bât. PSE-A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1015 Lausanne, Switzerland</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:Karl.Aberer@epfl.ch">Karl.Aberer@epfl.ch</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +41 21 693 4679</td>
</tr>
<tr>
<td>Essex County Council</td>
<td>Essex</td>
<td>Mary Rowlatt,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Essex County Council</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>United Kingdom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:maryr@essexcc.gov.uk">maryr@essexcc.gov.uk</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +44 (0)1245 436524</td>
</tr>
<tr>
<td>Forschungszentrum Informatik</td>
<td>FZI</td>
<td>Andreas Abecker</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Forschungszentrum Informatik</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Haid-und-Neu Strasse 10-14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>76131 Karlsruhe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Germany</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:abecker@fzi.de">abecker@fzi.de</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +49 721 96540</td>
</tr>
<tr>
<td>Company</td>
<td>Contact Information</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
| Institut für Informatik, Leopold-Franzens Universität Innsbruck (UIBK) | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
| ILOG SA                                     | Christian de Sainte Marie  
9 Rue de Verdun, 94253, Gentilly, France  
E-mail: csma@ilog.fr  
Tel: +33 1 49082981 |
| inubit AG                                   | Torsten Schmale, inubit AG, Lützowstraße 105-106  
D-10785 Berlin, Germany  
E-mail: ts@inubit.com  
Tel: +49 30726112 0 |
| Intelligent Software Components, S.A. (iSOCO) | Dr. V. Richard Benjamins, Director R&D  
Intelligent Software Components, S.A.  
Pedro de Valdivia 10  
28006 Madrid, Spain  
E-mail: rbenjamins@isoco.com  
Tel: +34 913 349 797 |
| NIWA WEB Solutions                          | Alexander Wahler  
NIWA WEB Solutions  
Niederacher & Wahler OEG, Kirchengasse 13/1a  
A-1070 Wien  
E-mail: wahler@niwa.at  
Tel.: +43 131 95843 11 |
| The Open University                         | Dr. John Domingue  
Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK  
E-mail: j.b.domingue@open.ac.uk  
Tel.: +44 1908 655014 |
| SAP AG                                      | Dr. Elmar Dorner  
SAP Research, CEC Karlsruhe  
SAP AG  
Vincenz-Priessnitz-Str. 1  
76131 Karlsruhe, Germany  
E-mail: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
| Sirma AI Ltd.                               | Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor  
135 Tzarigradsko Chausse, Sofia 1784, Bulgaria  
E-mail: atanas.kiryakov@sirma.bg  
Tel.: +359 2 9768 303 |
<table>
<thead>
<tr>
<th>Unicorn Solution Ltd.</th>
<th>Unicorn</th>
<th>Jeff Eisenberg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unicorn</td>
<td>Unicorn Solutions Ltd,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Malcha Technology Park 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jerusalem 96951,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Israel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +972 2 6491111</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>VUB</td>
<td>Pieter De Leenheer,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Starlab- VUB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vrije Universiteit Brussel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pleinlaan 2, G-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1050 Brussel, Belgium</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +32 (0) 2 629 3749</td>
</tr>
</tbody>
</table>
# Table of Contents

1 Motivation

2 Notion of Discovery
   2.1 Discovery Based on Semantic Annotation
   2.2 Different Phases of Service Usage

3 Discovery Framework
   3.1 General Architecture
   3.2 Distribution Mechanism
   3.3 Discovery Component
   3.4 External Components

4 API

5 Semantic Matchmaking
   5.1 Characteristics of Semantic Service Descriptions
      5.1.1 Design Decisions within the WSMO framework
      5.1.2 Relation to Parameter-Based Approaches
   5.2 Matchmaking Based on Abstract Service Descriptions
      5.2.1 Interpretation of Service Descriptions
      5.2.2 Notion of Matchmaking
      5.2.3 Object-Based Descriptions in the WSMO Conceptual Model
   5.3 Realising Matchmaking through Reasoning
      5.3.1 Matching Service Descriptions in Ontology Languages
      5.3.2 Matching Service Descriptions in WSML-Flight
      5.3.3 Matching Service Descriptions in WSML-DL
      5.3.4 Discussion of the Matchmaking Approach

6 Discovery in DIP Use Cases
   6.1 Telecommunications
   6.2 eBanking

A An ASM Specification of the Discovery Framework
   A.1 Formalization of SWS discovery framework
      A.1.1 Abstract Message Passing
      A.1.2 The SEND/RECEIVE submachines
   A.2 The PROCESSGOAL Submachine
   A.3 DISCOVERYENGINE ASM
1 Motivation

Semantic Web Services are envisioned as the enabling technology for the next generation of web applications. The objective of Semantic Web Services is to enable distributed computation over the Internet by automated and dynamic discovery, composition, and execution of services.

The automated and dynamic discovery is a crucial building block within the Semantic Web Services Technology Stack. Since the discovered web services have to be further processed (i.e., automatically selected, composed and executed) by an automated agent, the realisation of automated discovery requires discovery techniques providing matches of a very high quality. Especially, any ambiguities concerning “what the discovered service does” must be avoided.

The only state-of-the-art discovery approaches promising the required quality of match rely on reasoning over semantic annotations of web services. Semantic annotations of web services are expressed in terms of an ontological vocabulary. In the DIP project, the usage of the ontological vocabulary is regulated by the WSMO ontology for modelling service semantics and is expressed by the means of the WSML language.

The semantic reasoning is very expensive in the sense of required computational resources and cannot be performed in a distributed way. This may cause serious performance problems if large sets of web services have to be analysed. The only way to increase performance is to distribute the web service descriptions over several discovery locations and to perform the matchmaking of individual web services on different locations in parallel.
2 Notion of Discovery

In the vision of the Semantic Web, human users are replaced by computational agents that surf the web on their behalf, fulfilling some mission. For this purpose, the content and functionality offered in the web is being annotated by machine-interpretable semantic meta data. While human users focus on surfing through mostly static web content, agents call interfaces of service endpoints to invoke and consume their functionality. The semantic annotation of such endpoints tells the agent “what the service does”. A crucial step is for an agent to decide which service endpoint offers the relevant functionality it needs to fulfil its current mission. This is denoted as the problem of service discovery in the Semantic Web, or SWS discovery for short. Within the DIP project, semantic annotation of Web Services is formulated in terms of an ontological vocabulary that is aligned to the WSMO ontology for modelling service semantics, and is expressed in the WSML language as a knowledge representation formalism.

2.1 Discovery Based on Semantic Annotation

We call the annotation that specifies the semantics of a service a semantic service description, or service description for short. In general, such a semantic service description covers many different aspects of the service, ranging from the actual capabilities in some domain of value to the ontological grounding of service parameters at the message exchange level. In any of the phases of discovery, execution, composition, etc., different subsets of these aspects are taken into account. Since, in the first place, the finding of relevant service offers should be based on “what the service does” while details about how to communicate with the service play a minor role, we base our notion of discovery on the specification of the capabilities a service provides in its domain of value. In the context of discovery, we therefore understand a semantic service description also as semantic capability description. The domain of value of a service, as defined in [23], is the application domain associated to the service’s functionality, as e.g. travelling, eBanking, telecommunications, etc.

In the work that is carried out in the context of D4.17, the focus is set on discovering services according to Quality of Service (QoS) information. There, the authors have a slightly different notion of a service description - for details about the integration of QoS information into semantic service descriptions, we refer to D4.17.

Figure 2.1 depicts the role of semantic service descriptions in a generic discovery scenario. In this scenario, providers of services publish their service offers, whereas a requestor party issues a service request. They both refer to services in some domain of value. On the interface level, the providers implement Web Services that realise these services as a technical means to access their functionality and to make use of the value they provide. According to the Semantic Web idea, there is a layer of semantic annotation in between, on which a service description links a Web Service to a service in a domain of value: the service description annotates a Web Service interface and it describes the service in the domain of value which this interface provides access to.

The Web Services and semantic descriptions on the levels of interface and semantic annotation are computational objects that reside within the scope of the machine’s information space. Contrarily, the entities in the domain of value can not be directly processed within this information space but have their representation through the ontological de-
scriptions involved. Ontologies play a major role in describing the meaning of content in the Semantic Web. In our discovery scenario they have a twofold use: on the one hand the WSMO conceptual model is used as an upper-level service ontology to form the basis for modelers to express service semantics in terms of generic service-related constructs; on the other hand domain ontologies are plugged in to describe service capabilities in terms of the domain of value of a service. By making use of knowledge representation techniques, semantic capability descriptions try to capture “what a service does” in form of ontological descriptions.

The semantic capability descriptions serve as input for the discovery process, which compares the requestor’s description with those of providers to figure out which service offer is relevant for the request. This means that discovery operates on the ontological descriptions of the capabilities of a service rather than on specifications of Web Service interfaces. As a result, the discovery process returns references to service descriptions considered relevant, together with references to their associated Web Service interfaces. The requestor agent has then the possibility to further investigate the relevant service offers by either looking at their semantic descriptions in more detail or by directly calling the Web Service interface.

2.2 Different Phases of Service Usage

To complement our notion of discovery, we characterise the discovery phase within the lifecycle of service usage. According to [23], the relationship between a requester and a provider party goes through three different phases: a service discovery phase, in which potential providers are discovered; a service definition phase, in which the concrete service to be carried out is defined in all its details; and a service delivery phase, in which the value of this concrete service is actually delivered to the requester. Similarly, the lifecycle

\[1\] Again, for a description of how QoS information is taken into account, we refer to D4.17.
presented in [17] consists of a goal discovery phase, taking into account that it needs some effort to come to a “goal description” that properly expresses the service request, a Web Service discovery phase and a service discovery phase, which map to the discovery and definition phases in [23].

Based on these lifecycle models, we consider the following four phases to position our notion of discovery.

1. **Goal Definition** concentrates on finding a predefined semantically described WSMO goal in a repository, which can be refined by the service requestor in order to be used as a search criterion for service discovery. Goal definition has been introduced because it is assumed that in general a user issuing a service request will not be faced with the machine-interpretable representation of a WSMO goal in the WSML language. Instead, such a user will probably be provided with a graphical user interface for browsing a repository of goals, or with some intuitive input mask for refining goals. The techniques that are used to search for predefined goals and to refine them probably comprise keyword-based retrieval and navigation in ontologies. The abstract goals from the repositories must be in general adapted to user needs, e.g., specialisation of concepts, restriction of property values etc. The result of this phase is a formally specified goal that can be used as a search criterion for service discovery.

2. **Service Discovery** is based on comparing the semantic description of a requested abstract service against those of provided abstract services in terms of relevance. According to [23], an *abstract service* is distinguished from a *concrete service* in that the former abstracts from the concrete service parameters which determine the latter. Thus, an abstract service describes a class of service parameter configurations. In the WSMO conceptual model, the semantic service description for the requested abstract service coincides with a WSMO *goal* element, whereas the semantic descriptions of provided services coincide with WSMO *web service* elements. Service discovery is performed by matching the goal against available web services. Notice that input to the service discovery phase are the semantic descriptions of goals and web services only, which are contained in their associated WSMO *capability* elements. This implies that the matching is solely based on information contained in these descriptions of abstract services. A selection of a service due to concrete parameter information, which possibly requires invocation of the services interface, is in the scope of the subsequent service definition phase.

3. **Service Definition/Selection** starts from an already identified set of potential WSMO web service candidates which have been identified as relevant for a goal in the service discovery phase. It possibly involves negotiation of service parameters, and thus, invocation of the candidates Web Service interface. This might also involve protocol mediation when the requested interface does not fit the provided interface. Figuring out which service to finally choose is beyond the information contained in the semantic descriptions of goal and Web Service elements. In this phase, the generalisation to abstract services is given up and a concrete service with a concrete parameter configuration has to be defined, as it is later on delivered. Successful service discovery does not necessarily lead to successful delivery of a service, since in the set of potential service candidates there might be no one that is finally able to define a concrete service on which both the requestor and provider agree.

4. **Service Delivery** comprises any steps required for the actual invocation and consumption of the concrete service selected for execution. This involves the invocation
of the Web Service interface according to its choreography as well as different forms of mediations concerning the protocol or the data to be transmitted.

To complete our notion of discovery, we see the focus of generic tool support for discovery to be set on the phase of service discovery. In this sense, the DIP discovery module should support a Semantic Web agent in finding service providers which potentially meet its requested service requirements, based on the semantic descriptions of service capabilities.

The phase of goal definition seems to be too specific to the respective application for providing generic solutions to create or retrieve goals in a discovery framework.

The phase of service definition/selection would require a tight integration with work on choreography and invocation of Web Service interfaces and breaks the scope of processing semantic descriptions. Ideas that include negotiation of parameters according to preference information would require an extra encoding of preferences which is currently not present in the WSMO conceptual model or semantic service descriptions in general. Furthermore, such a negotiation is also very specific for the domain of value of the service, such that generic tool support is difficult to achieve. We argue that the main actor in the service definition/selection phase is the agent invoking the interfaces of potential Web Service candidates that have been identified as such during discovery.
3 DISCOVERY FRAMEWORK

3.1 General Architecture

A high level architecture of a service discovery component is illustrated in Figure 3.1. A user sends a goal (initial goal defined in the goal definition phase) to the SWS discovery service and expects as a result a set of Web Service descriptions matching this goal. This set of Web Services has to be further processed by the agent of the service requester in the service definition phase. An SWS discovery service may be distributed over several search locations. In that case, the SWS discovery service has a search distribution mechanism identifying external discovery locations and querying them as well as the local search location with the goal as a search parameter. The semantic matchmaking takes place at each queried location locally. The results are collected and aggregated by the requesting discovery locations and finally delivered to the initial requestor.

The architecture of the discovery component takes the discovery distribution into account. However, the main focus will be on semantic discovery, i.e., in principle the distribution framework can be refined in order to support sophisticated distribution concepts. In DIP, however, we will work on a centralised repository or allow distribution for simple load balancing only.

The framework for the search distribution mechanism, the semantic discovery component and the relationship to external components are described in more detail in the next subchapters.
3.2 Distribution Mechanism

The distribution of the semantic Web Service discovery process is motivated by the fact that semantic matching of a goal and a Web Service capability description is expensive in the sense of the required computational resources. The distribution of Web Service descriptions over several search locations allows to match a goal with the set of Web Service descriptions locally available at the respective search location independent of other search locations. Therefore, the local sets of Web Service descriptions matching a goal can be calculated in parallel at different search locations reducing herewith the total query execution time.

The simplest form of distribution is the load balancing storing equally large sets of Web Service descriptions at each search location. However, more sophisticated approaches taking into account either some service classification taxonomies or analysing ontologies and concepts used in the Web Service descriptions could provide more intelligent distribution strategies. Such distribution strategies could help not only to parallelise the processing of a goal query but also to reduce the total number of Web Service descriptions to be matched with a goal, e.g., by disclosing search locations extraneous to the goal from search.

A generic framework for a search distribution mechanism is illustrated in Figure 3.2. This framework has to be further refined in order to implement a concrete distribution strategy. The following basic principles must be fulfilled independent of the deployed distribution strategy:

- The distribution mechanism must guarantee that all relevant discovery locations are queried during the global search (Identification and Querying of WS Discovery Locations).
The distribution mechanism should avoid querying of not relevant discovery locations in order to minimise the number of queried discovery locations (Identification and Querying of WS Discovery Locations).

The distribution mechanism must be able to detect search loops and terminate search for duplicate requests coming from external discovery locations (Break Condition).

The distribution mechanism has to wait until search results from all requested search locations have arrived or a timeout occurs (Collect Search Results).

A high-level Abstract State Machines (ASM) specification formalising the introduced distributed discovery framework is provided in Appendix A.

Figure 3.3 illustrates the idea of the distribution of a goal query in the proposed architecture.

Figure 3.3: Query distribution

3.3 Discovery Component

The internal architecture of the local discovery component is depicted in Figure 3.4. A local discovery location performs two steps in order to match the goal with the locally available Web Service descriptions. In the first optional step a (Non-)semantic pre-filtering reduces the set of Web Services to be semantically matched in the second step Semantic Matchmaking.

The filtering procedures applied in the first step must guarantee that no potentially matching Web Services are filtered out. Possible techniques to be used in the first step are keyword-based matching and navigation in classification hierarchies for services.
In the second step, the set of Web Services is matched against the goal and the resulting set of possibly ranked matching Web Services is delivered to the distribution mechanism collecting the search results. Both steps may require an external ontology mediation service for the reasons already described above.

Optionally, the set of services matching the goal can be further reduced by matching non-functional QoS properties of the goal and (static) QoS properties contained in the Web service descriptions. A static QoS property is assigned by a service provider directly to its Web Service description(s) and does not change until the Web Service description is updated through the service provider. Dynamic (taking into account user feedback and/or monitoring of the Web Service behaviour) QoS discovery will be solved by an additional solution provided by EPFL[15]. A module supporting evaluation of static QoS properties may make sense if the proposed discovery solution is deployed stand-alone (without the solution described in [15]). As for the semantic matchmaking, the exact method according to which the semantic descriptions of goals and Web Service capabilities are compared is described in Chapter 5. A set of concrete discovery examples comprising Web Service capabilities and goals modelled in WSML from the different case studies can be found in chapter 6.

3.4 External Components

As already illustrated in Figure 3.4 there is a relationship to at least two external components: Ontology Mediation Service and a registry or repository storing ontologies and Web Service descriptions. The relationship to the storage containing the ontologies and
Web Service descriptions is obvious and will not be discussed further. The relationship to a mediation service requires more attention.

One could assume that Web Service capabilities and goals are described using the same terminology. Then the ontology mediation problem does not exist during the discovery process. However, it is unlikely that a potentially huge number of distributed and autonomous parties will agree before-hand on a common terminology.

Alternatively, one could assume that goals and Web Services are described in terms of 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 a scenario relies on three main assumptions:

- Goals and Web Services most likely use 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 controlled vocabularies or ontologies to describe requested and provided services.
- There is some ontology 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. A mediator should provide an integrated view on such mediated ontologies, make them appear as one global view.

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 ontology mediation support that is a prerequisite for successful discovery. Therefore, with respect to mediation, service discovery requires only mediation on ontologies and their instances.
4 API

The semantic discovery component does not live in an isolated environment. It has, therefore, to take into account compatibility requirements of several external components it interacts with. It must communicate with the outside world in terms of goals and service capabilities as specified in WSMO. It must be compatible with the WSMX discovery interface for compatibility reasons of the DIP architecture. Semantic discovery and dynamic QoS-based discovery (ref to EPFL solution) must offer at least the same core discovery interfaces allowing to use these components in the same fashion either as stand alone solutions or within an integrated QoS-enabled semantic discovery framework. Therefore, the API of the semantic discovery component has to comply with several requirements:

- Goals and Web Service descriptions exchanged over discovery interfaces have to be compatible with the WSMO conceptual model and have to be expressed in the WSML language.
- The discovery interface has to be compatible with the WSMX discovery interface.
- The core discovery interface has to correspond with the QoS-based discovery component.
- The discovery interface has to be specified either in WSDL allowing remote access to the discovery component over a Web Service interface or in Java in order to allow direct local integration with WSMX environment.

Taking into account the above requirements the discovery interface provides the following functionality:

\[
\text{Discover(In: Goal,}
\text{Out: List of WS-description)};
\]

This method specifies the core discovery interface for interacting with the discovery component. It takes as input a WSMO Goal and provides as output a list of WSMO WS-descriptions matching the goal.

\[
\text{Discover(In: Goal,}
\text{[In: RankingOntology],}
\text{Out: List of WS-description)};
\]

This method is introduced for interoperability reasons with the WSMX discovery interface. It takes as input a WSMO Goal and a RankingOntology\(^1\) and provides as output an ordered (ranked) list of WSMO WS-descriptions. WSMX does not specify what a ranking ontology is or how to use it for ranking of results. Therefore, this specification does not elaborate further on this method and considers it as a placeholder for potential future work.

\[
\text{Discover(In: Goal,}
\text{[In: RankingOntology],}
\text{Out: List of \{WS-description, InterfaceID\}}});
\]

This method is introduced for interoperability reasons with the QoS-based discovery interface. The semantic discovery component does not make any use of the RankingOntology provided as input and behaves as if the Discover method of the core interface has been called. The functionality of this method is described in the QoS-based discovery specification.

\(^1\)Square brackets in the method description indicate that the parameter RankingOntology is optional.
5 Semantic Matchmaking

In Section 2 discovery is introduced as the process of identifying service providers whose offers are relevant for a service request, based on semantic service descriptions. The technique we employ for comparing semantic descriptions of providers and requesters in terms of relevance is semantic matchmaking. The overall discovery process has to take into account all the available service offers, whereas in the context of semantic matchmaking we only consider a single offer together with the request. The matchmaking process has to decide whether this offer is relevant for the request by looking at the semantic descriptions of both. Since such descriptions are expressed in terms of the WSMO ontological model using the WSML knowledge representation language, semantic matchmaking is based on the application of logical inferencing as an automated reasoning technique.

5.1 Characteristics of Semantic Service Descriptions

A discovery framework that employs semantic matchmaking basically needs to specify the following two essential things:

- how semantic descriptions of services are modelled, and
- how semantic descriptions are compared in terms of relevance.

These two aspects are intertwined with each other and for both of them a discovery framework must define precise methods and the components to support these methods. Defining the relevance comparison mechanism alone is not sufficient, since it is then not clear what kind of descriptions serve as input. Instead, the comparison method must take into account the specifics of the chosen way of modelling and the modelling method must be suitable for the chosen way of how comparison is performed.

5.1.1 Design Decisions within the WSMO framework

The WSMO framework gives some indication of how the functionality of a service is encoded in WSMO-capability elements. In [17], however, it also leaves some final decisions of what exactly is expressed through such WSMO-capabilities open, supporting different alternatives for discovery approaches. One of the alternatives is to model service capabilities as transitions of states describing the service before and after its execution, respectively. Another alternative is to characterise capabilities by means of the “objects” a service delivers in its domain of value [17].

Starting from [17], we aim towards an object-based approach for describing service semantics instead of one based on descriptions of state-transitions, since this seems to better fit the abstract services in the discovery phase as defined in Section 2. We argue that this way of modelling service semantics is closer to the user’s intuition in the light of the DIP use case scenarios. In such an object-based description approach the functionality of a service is expressed in terms of the “objects it is able to deliver”, modelled through concepts and relations in an ontology language. Contrarily, in a state-based description approach both the pre-state and the post-state with respect to service execution would be modelled explicitly.

This notion of an “object delivered by the service” is somewhat abstract and its description depends on the way in which the domain of value of the service is modelled. Table 5.1.1 shows some examples of what such “objects” could be in different domains.
of value. In the eBanking use case from WP10, for example, the domain is described

<table>
<thead>
<tr>
<th>domain of value</th>
<th>“object to be delivered”</th>
</tr>
</thead>
<tbody>
<tr>
<td>eBanking WP10</td>
<td>mortgage loan</td>
</tr>
<tr>
<td>Travelling</td>
<td>ticket</td>
</tr>
<tr>
<td>Logistics</td>
<td>shipping contract</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 5.1: Example “objects to be delivered by services” in different domains of value.

by financial ontologies, hence a modelling decision was to describe the functionality of a mortgage service (see WP10 documentation and Section 6.2) in terms of the “mortgage loan object” it delivers. In the travelling domain the decision could be to describe the ticket that specifies the details of a journey offered by a travel agency, whereas in the logistics domain the shipping contract could serve specification of capabilities.

A similar description approach was also followed in the former project SWWS (IST-2001-37134). We could reuse some of the ideas introduced there and adapt them to fit into the DIP setting.

In the light of the notion of an abstract service, the “object to be delivered by a service” is described by an ontological vocabulary consisting of concepts and relations from the domain of value of the service. Describing these “objects” by means of ontological concepts allows modelers to express variability in their descriptions: instead of exactly specifying all the parameters and property values of a concrete such object, the description of an object class comprises a multitude of concrete instantiations. In the sense of this variability, a semantic description of an abstract service is associated with a set comprising all accepted parameter configurations of what the service is able to deliver. (We refer to [14] for details on this variability.)

Figure 5.1 depicts the decisions taken for the chosen approach of modelling service semantics with respect to semantic matchmaking.

![Figure 5.1: Decisions for a modelling approach. Highlighted arrows indicate taken decisions, whereas dashed arrows indicate alternatives.](image-url)
Motivated by the Semantic Web idea of providing machine-interpretable annotation for services mentioned in Section 2, we first restrict the scope to representing “what the service does” in an abstract way. Within this scope, we opt for the above mentioned object-based approach when describing service functionality. Further, we represent the “object to be delivered” by means of a class of objects rather than by direct instances. In the WSML language such a class corresponds to a concept. On the different levels, the elements used to describe service semantics are referring to elements used to describe the domain of value of the service, captured in domain ontologies.

5.1.2 Relation to Parameter-Based Approaches

To further characterise our chosen object-based description approach, we relate it to other approaches outside the WSMO framework that focus on the semantic tagging of input and output parameters of a service. These notions can, for example, be found in WSDL-S [2] descriptions but also in the OWL-S [1] service profile. We sketch the difference to these approaches in Figure 5.2.

![Figure 5.2: Different approaches of describing service semantics.](image)

Both types of approaches to capture the semantics of a service / Web Service aim to use domain ontologies that are modelled independently from their usage in discovery or even service description. The ontology fragment in Figure 5.2 covers terms around a travelling service scenario, such as “Ticket”, “Airplane”, “City”, “source” and credit card information.

In approaches based on the semantic tagging of Web Service interfaces, shown in a), the annotation of a Web Service directly refers to the elements that occur in this interface, i.e. input and output parameters. Each parameter is lifted to the ontological level and mapped to some element in the ontology, e.g. to a concept. In this way, the machine “sees” a credit card number no longer as a mere integer but as a concept for which some ontology assigns meaning by a proper axiomatisation and relation to other concepts. For
discovery and matchmaking, two such semantically lifted interface descriptions match when their input and output parameters are compatible.

In approaches based on an upper-level service ontology like WSMO, shown in b), the semantic annotation of a Web Service refers to some service related entities, such as a WSMO capability, that abstract from the level of input and output parameters. As introduced in Section 2, the Web Service is just the technical means to “realise” some service in a domain of value. In our approach, semantic annotation describes this service or the “object it delivers”, respectively. Hence, as shown in Figure 5.2 b), the service/object is semantically mapped to some concept in the domain ontology as a whole. This allows for making statements about the service/object as a whole on the ontological level.

In comparison to the parameter-based approaches in a), we consider the approaches in b) as more powerful in terms of expressiveness in semantic annotation. Mirroring the notion of a service itself on the ontological level allows to better capture the service’s functionality and to better express what this service actually does. Contrarily, by just referring to input and output parameters it is sometimes difficult to capture the service functionality. Consider, for example, one service for checking whether a credit card is still valid and one for cancelling a credit card to avoid abuse. Both have the same input parameter, say, the credit card number. By just lifting this number to the ontological level the distinction between the two services is not captured in terms of functionality. Their interfaces would be deemed compatible while their semantics is not.

5.2 Matchmaking Based on Abstract Service Descriptions

As mentioned above, semantic matchmaking involves two service descriptions: one advertised by a service provider and one issued by a requester. Both the requester’s and the provider’s description describes the “object to be delivered by the service”, but from different points of view. While the requester describes the kind of “object” he would like to consume, the provider describes the “object” which his service is actually able to deliver. On the other hand, by using object classes both parties employ variability in their descriptions, allowing for a multitude of concrete service objects requested or provided, respectively. Hence, the intention behind semantic service descriptions is to express alternatives of accepted concrete services.

5.2.1 Interpretation of Service Descriptions

An example from the travelling domain is depicted in Figure 5.3. Suppose a travel agency offers tickets for travelling within Europe by airplane, train or bus. As indicated in Table 5.1.1, a modelling decision taken here is to express the capability of an online ticket booking service through the “ticket object” such a service delivers.

For the moment, we abstract from WSMO descriptions specified in the WSML language, and so the semantic description sketched in Figure 5.3 expresses the capability of the ticket booking service intuitively. In a non-formal way it says that the object delivered by the service is a ticket whose source and destination locations are in Europe and which can be issued for either airplane or ground vehicles. These constraints can be met by a variety of different tickets, and whenever a customer requests a ticket within these constraints the travel agency will agree on selling it to the customer. It will not agree if the requested ticket lies outside these bounds, say when asking for a trip to USA or for an
ocean cruise. Hence, the service description represents the set of all concrete objects (here they are tickets) which the provider of the service would agree on delivering. Further, all these different concrete objects are interpreted as alternatives of service delivery – in any alternative all the details of an object’s parameters are specified.

Symmetrically to the service offer description in Figure 5.3, a description issued by a service requester represents the set of concrete objects which the requester is willing to agree on being delivered.

We highlight again that our service capability descriptions specify abstract services, meaning that the modeler of such a description abstracts from what a service can actually deliver in reality. In general, they describe an approximation of the service’s behaviour to avoid overloading the semantic specification. This implies that for some cases they give wrong information about what the service is able to deliver. In the sense of [23], our service descriptions are therefore complete but not correct, meaning that they capture all the concrete services that can be delivered but also some which cannot. Some of the information needed to determine whether a particular concrete service can be delivered is dynamic and cannot be included in static semantic annotation. As an example, consider a travelling service that offers train tickets but, at a certain point in time, a particular train might be booked out and further requests are refused – this cannot be captured by the semantic description and is subject to the service selection phase (see Section 2).

5.2.2 Notion of Matchmaking

Having defined how semantic descriptions of abstract services have to be interpreted, we can now describe how semantic matchmaking operates on such descriptions. Since the service descriptions of the requester and provider represent the concrete objects they agree on being delivered, the basic idea behind our notion of matchmaking is to check whether they have objects in common. As alternatives of service delivery the concrete objects can be seen as service parameter configurations. An offer is relevant for a request if the service can be configured such that both the provider and the requester agree on this configuration, which is represented by some concrete object to be delivered. In this case at least one of the concrete objects offered by the provider is in the scope of the request, which is sufficient to establish a positive match. This notion of matchmaking is illustrated in Figure 5.4.
Semantic Matchmaking checks for an intersection of the sets of concrete objects associated with the semantic descriptions of the requester and provider of the service. If there is an overlap between the two sets then the offer is relevant for the request. If the two sets are disjoint then the offer is not relevant, meaning that nothing of what the service is able to deliver is actually requested. Consider again the travel agency from the former example together with a request asking for a flight from Hamburg to Galway. As depicted in Figure 5.4, among all the tickets which the provider’s service can deliver there is one meeting these requirements, carried out on a Boeing 747 machine. Since the requester did not constrain on the type of machine that is used for the flight, there is another one intersecting, say, using a different machine. Other tickets, say for ship journeys, are not supported by the provider, whereas others, covering different cities, are not in the scope of the request.

5.2.3 Object-Based Descriptions in the WSMO Conceptual Model

One of the insights gained in the work on discovery in WP4 is that the object-based description approach, as an alternative specified in [17], does not entirely fit with the WSMO conceptual model. The WSMO-capability elements, designed to semantically express the functionality of a service, are more tailored to a description approach based on state transitions, for which they provide precondition, postcondition, assumption and effect elements. The kind of information we specify in our service capability descriptions, namely “what a service as a whole does or delivers”, does not seem to fit in any of these notions provided by a WSMO-capability, at first. Instead, it abstracts from state transitions and describes the service in a more holistic way, rather than describing the single pre- and post-states.

Hence, one of our design decisions is to see the different fields of a WSMO-capability expressing the semantics of a service conjunctively. All of them can be used to capture parts of our semantic service descriptions, however, the discovery process described here does not take into account their role of describing pre- and post-states. Although, the modeller might distinguish between things that hold before and things that hold after service execution, respectively, for other purposes.

The differentiation between conditions and assumptions/effects is meant to distinguish between an information space, which is a model of what is computationally processed...
within the machine, and the real world, in which the service takes effect. This explicit
distinction allows for making statements about the service as computational entity, as
well as about the real world service counterpart, separately from each other in ontological
descriptions. Since, for our discovery process, we propose to describe service semantics in
the (rather abstract) model of the domain of value of the service, we interpret ontological
service descriptions in terms of statements about the real world. Therefore, our discovery
process does not take into account an explicit distinction between pre- and postcondition
on the one hand and assumption and effects on the other hand.

Finally, this means that the information in a semantic service description, as defined
here, is directly mapped onto a WSMO-capability, possibly spread over the different fields
it provides (in the sense of taking the union of all logical expressions in the different slots.)

Alternatively, one could think of a proposal for extending the WSMO conceptual
model to better capture the object-based capability descriptions of the before mentioned
kind.

Other approaches to modelling service semantics base the semantic description of a
service’s capabilities on the notion of a service task. Such a notion describes a service
as some kind of activity and can be found e.g. in [12], work that has been carried out
in the scope of the WonderWeb project (IST-2001-33052), and also in [23]. In alignment
with the distinction between information space and real world descriptions, the notion of
a service task semantically captures some action that is performed when a Web Service
is invoked or when a service is being delivered, respectively.

The properties of a capability, namely preconditions, postconditions, assumptions and
effects, could equally well be seen as properties of such an action or task associated to the
service. Furthermore, the conceptual model could facilitate the instantiation (or subclassing)
of such a service task itself, inheriting the conditions and assumptions/effects, but
also allowing for custom properties. Contrarily, the way a WSMO-capability is currently
used is to instantiate its properties by logical expressions – they are filled with semantic
descriptions that make statements about the pre- and post-state, intended to express the
service’s capability, but no statements about the capability as an ontological element can
be made.

When adopting this notion of a service task, the “classes of objects delivered” that
we describe in our approach could probably be aligned with the service task more easily
than they can with notions of state transitions. Moreover, in terms of methodological
guidelines of how to formulate service descriptions, our approach would also benefit from
the richer axiomatisation proposed in the work around [12], which clarifies the relation
between the elements in a top-level service ontology.

In the context of QoS-enabled discovery and trust there are other suggestions to extend
the WSMO conceptual model, documented in D4.17 and D3.6.

5.3 Realising Matchmaking through Reasoning

In this section we show how the process of matchmaking between semantic service de-
scriptions is realised by making use of reasoning techniques. In this way, we link the work
on discovery in WP4 to reasoning in WP1.

We first introduce a formalisation of service descriptions and their matchmaking in
terms of ontology languages in general. In this way we abstract from concrete knowledge
representation formalisms like the WSML language. In the subsequent sections, we then
map this formalisation to two variants of this language, namely WSML-Flight and WSML-DL.

5.3.1 Matching Service Descriptions in Ontology Languages

Ontology languages in general provide constructs for describing concepts, their instances and relations between them - these are the basic elements ontologies usually are built of. In languages inspired by first-order logic formalisms, instances are often seen as individuals (objects) of some universe of discourse and instance-relationship holds between concepts and individuals. A concept groups together a set of individuals, its instances, that share common characteristics.

Relations are often restricted to binary relations, represented as sets of pairs of individuals, and are also referred to as properties. The properties of a concept are the relations that hold between the instances of this concept and other individuals. (In WSML properties are also called attributes.)

Additionally, most ontology languages also support the handling of datatypes, such as integer or string, and their values. The properties of a concept can range over other concepts or over some datatype.

Moreover, many languages allow to describe concepts in terms of restrictions that constrain their sets of instances. This is done by either restricting their scope to other concepts, e.g. by subsumption, or restricting their properties.

Formalisation of Service Descriptions

Recall from section 5.2, that, in our service descriptions, we want to express abstract services and the “classes of objects they deliver”. We map such a class of objects to a concept, which potentially allows for several instances. In this sense, the concept expresses information about the abstract service, whereas its instances express information about all the intended concrete services captured.

We denote by $S$ the concept that represents the class of objects delivered by the service. A service capability description describes this concept $S$ by means of statements in some ontology language, referring to domain ontologies that provide a vocabulary of the domain of value of the service. We also call $S$ the service concept, since it is associated with the service to be described. Similar to the approach presented in [13], we describe the concept $S$ in terms of restrictions that constrain the intended concrete services.

Semi-formally, a description of the concept $S$, as motivated by example in Section 5.2, aggregates a set of restrictions as follows:

$$S \simeq C_1, C_2, \ldots, C_m$$

$$\frac{p_1}{R_1} \frac{P_2}{R_2} \ldots \frac{p_n}{R_n}$$

$$R_j \simeq C_{R,j1}, C_{R,j2}, \ldots, C_{R,jm}$$

$$\frac{PR_j1}{R_{R,j1}} \frac{PR_j2}{R_{R,j2}} \ldots \frac{PR_jn}{R_{R,jn}}$$

$$S \simeq \text{Ticket}$$

$$\frac{from}{\text{Europe}} \frac{to}{\text{Europe}} \frac{vehicle}{Airplane_1}$$

$$\frac{\text{Airplane}_1}{seats} \geq 50$$
The description restricts $S$ by concepts $C_i$ and by properties $p_j$ for which it specifies ranges $R_{ij}$. Depending on whether a property ranges over a datatype or over individuals, the ranges $R_{ij}$ are datatype intervals or concepts, respectively. In the example (on the right-hand side), $S$ is restricted to ticket objects within Europe issued for flights (airplanes).

If the ranges $R_{ij}$ are concepts, they can further be restricted themselves as part of the description. In the example the (newly introduced) concept $Airplane_1$ is restricted to airplanes having more than 50 seats.

To capture this formalisation of a description for $S$, we define two different types of restrictions $\rho$,

$$\rho(C_a, C_b) \; ,$$

restricting a concept $C_a$ to the scope of a concept $C_b$, and

$$\rho(C, p, R) \; ,$$

restricting a property $p$ of a concept $C$ to a range $R$. Building on the description given above, we recursively define a set of restrictions for a concept $C$ as

$$R_C = \{\rho(C, C_1), \ldots, \rho(C, C_m), \rho(C, p_1, R_1), \ldots, \rho(C, p_n, R_n)\} \cup \bigcup_{1 \leq j \leq n} R_{R_{ij}} \; ,$$

such that it contains restrictions for all involved concepts which the modeller intends to put constraints on.

A description of a service concept $S$ in an ontology language is then just the set of restrictions $R_S$. Within the restrictions $\rho$ in $R_S$ a modeller refers to concepts, properties and individuals defined in domain ontologies. In the example, properties like from or seats and concepts like Ticket or Airplane most likely stem from some ontology about travelling, whereas the concept $Airplane_1$ has been introduced within the service description.

**Formalisation of Matchmaking**

In the process of semantic matchmaking three different sources of knowledge are involved: the service description of a requester, the service description of some provider and the domain knowledge the two are referring to. Both of the service descriptions specify concepts, $S_r$ for the requester and $S_p$ for the provider respectively, and are represented by sets of restrictions $R_{S_r}$ and $R_{S_p}$. Since the above introduced restrictions $\rho$ are expressed in form of statements in an ontology language, the sets $R_{S_r}$ and $R_{S_p}$ can be interpreted as sets of facts in some knowledge representation formalism on which we can perform reasoning.

The domain knowledge comes in form of domain ontologies that provide vocabularies relevant for the domain of value of the services described, and which requesters and providers refer to in their descriptions. We denote by $O$ the set of such relevant ontologies.

For reasoning, we need to include all these pieces of knowledge into a knowledge base $KB$, defined as follows:

$$KB := R_{S_r} \cup R_{S_p} \cup \bigcup_{O_i \in O} O_i \; .$$
The knowledge base $\textit{KB}$ is build temporarily for the reasoning process and is dismissed afterwards. In this sense, it is dynamic knowledge, put together for each matchmaking step.

The descriptions of the service related concepts $\textit{R}_s$ and $\textit{R}_p$ are designed by modelers on the side of the requester and provider parties, respectively, and make up the semantic service descriptions. They are particularly build for specific tasks around Semantic Web Services, like discovery, and are not used out side the scope of these tasks. In this sense, they are static within the use in tasks like discovery, in which they are repeatedly used. However, they are “less static” than the sources of domain knowledge which they refer to.

The domain ontologies in the set $\textit{O}$ are shared vocabularies in the context of the domain of value of the service – in our travelling example they cover e.g. geographic or transportation related knowledge. From the discovery point of view, they are static sources of knowledge which are consulted during the matchmaking process, but which are also used outside the scope of discovery. They have been built as reusable domain vocabularies and do, in general, not contain any discovery specific modelling.

For realising the matchmaking process by means of reasoning services, we define a matching condition $\mu(C_a, C_b)$ on concepts, which is true iff the two concepts $C_a$ and $C_b$ have a “sufficient degree” of overlap. According to the notion of intersection used in Section 5.2, this degree of overlap would be such that it is sufficient when the two concepts can potentially have some individual as a common instance, i.e. that they are not disjoint. To realise intersection, we will focus on this particular matching condition and denote it by $\mu_\cap(C_a, C_b)$. In [22] there have been proposed other forms of matching conditions requiring stronger degrees of overlap, that have been subsequently used in [20, 21, 17, 16] for establishing different degrees of matching between service descriptions.

We also take into account these other conditions in our formalisation and list all the matching conditions in the following Table. In terms of sets of intended concrete services,

\[
\begin{array}{|c|l|c|}
\hline
denotation & intuition & order \\
\hline
\mu_\cap(C_a, C_b) & concept intersection & 1 \\
\mu_\subseteq(C_a, C_b) & concept subsumption (specialisation) & 2 \\
\mu_\supseteq(C_a, C_b) & concept subsumption (generalisation) & 2 \\
\mu_\equiv(C_a, C_b) & concept equivalence & 3 \\
\hline
\end{array}
\]

Table 5.2: Matching conditions with different degree of overlap.

$\mu_\cap$ and $\mu_\subseteq$ check if one of the sets is fully contained in the other, whereas $\mu_\equiv$ checks whether the two sets coincide. The order number indicates how strict a condition is: if a condition with a higher number holds between two concepts then any condition with lower number also holds.

Another approach that addresses ranking based on structural difference operations for concepts is described in [9].

The reasoning process that realises matchmaking involves the knowledge base $\textit{KB}$ and the two concepts $S_r$ and $S_p$. We define it by the following consequence operation on a knowledge base and a matching condition

\[
\textit{KB} \models \mu(S_r, S_p) ,
\]

which evaluates whether the matching condition follows from the knowledge included in $\textit{KB}$. It can be seen as form of “intuitive entailment”. If this entailment holds then...
there is a positive match between the requester’s and the provider’s service descriptions, according to the matching condition $\mu(S_r, S_p)$ – otherwise there is a negative match. In the realisation of matchmaking with concrete WSML language variants we consider different formalisms defining different reasoning services, and therefore $\models$ does not necessarily coincide with entailment $|=\ $in the underlying formalism. Later, we map the “abstract” reasoning step in (5.1) to concrete reasoning services in WSML-Flight and WSML-DL.

5.3.2 Matching Service Descriptions in WSML-Flight

In this section we map the general formalisation of service descriptions to the WSML-Flight ontology language and show how matchmaking can be realised by applying the WSML-Flight reasoner delivered with D1.9.

The Flight variant of the WSML language, see D1.7 and [19] for details, is an ontology language that builds on a rule-based knowledge representation paradigm and is very similar to F-Logic [18]. Its semantics [D1.7] is defined such that any fragment of conceptual syntax, involving concepts, their instances and attributes (properties), is transformed into logic programming rules and interpreted under a minimal model semantics [10] in logic programming style. Hence, WSML-Flight formulas are interpreted under closed-world assumption.

The elements of WSML-Flight ontologies coincide with logic programming rules and sets of such rules coincide with logic programs. In this sense, the sets $R_{Sr}$ and $R_{Sp}$ and the ontologies $O$, in the WSML-Flight case, are seen as logic programs.

We define the mapping $\tau_{lp}$ for transforming concept and property restrictions, respectively, into WSML-Flight logical expressions in the form of logic programming rules in the following table.

<table>
<thead>
<tr>
<th>Restriction</th>
<th>WSML-Flight formula (rule syntax)</th>
<th>conceptual syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{lp}(\rho(C, p, R))$</td>
<td>$y , \text{memberOf} , R := x , \text{memberOf} , C \text{ and } x[p , \text{hasValue} , y]$</td>
<td>$C , p , \text{impliesType} , R$</td>
</tr>
<tr>
<td>$\tau_{lp}(\rho(C_a, C_b))$</td>
<td>$x , \text{memberOf} , C_b := x , \text{memberOf} , C_a$</td>
<td>$C_a , \text{subConceptOf} , C_b$</td>
</tr>
</tbody>
</table>

Table 5.3: Transformation of restrictions to WSML-Flight formulas.

Here, and in the following, $x$ is a universally quantified variable and $\iota_p$ is a new individual, not previously known to the knowledge base. In the table we also give the WSML conceptual syntax for the restrictions. Notice, that a property restriction maps to two rules, the second of which cannot be expressed in conceptual syntax – it serves the instantiation of “dummy facts” for the reasoning process, which is needed due to the closed-world semantics. (Notice also, that the second rule is problematic for being applied to datatype ranges because it is not clear which data value to choose for instantiation, which is discussed in Section 5.3.4.)

The transformed sets $\tau_{lp}(R_{Sr})$ and $\tau_{lp}(R_{Sp})$, i.e. application of $\tau_{lp}$ to all restrictions in the sets, are WSML-Flight ontologies (or logic programs) and build up the knowledge base $KB_{lp}$ together with WSML-Flight domain ontologies $O$ as follows

$$KB_{lp} := \tau_{lp}(R_{Sr}) \cup \tau_{lp}(R_{Sp}) \cup \bigcup_{O_i \in O} O_i$$
We extend the mapping \( \tau_{lp} \) to also capture the abstract entailment \( KB_{lp} \models \mu(C_a, C_b) \) of matching conditions, transforming it into WSML-Flight reasoning steps, in the following table.

<table>
<thead>
<tr>
<th>Matching Condition</th>
<th>WSML-Flight reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_{lp}(KB \models \mu_{\pi}(C_a, C_b)) )</td>
<td>( KB_{lp} \cup i \text{ memberOf } C_a \text{ and } i \text{ memberOf } C_b \text{ sat.} )</td>
</tr>
<tr>
<td>( \tau_{lp}(KB \models \mu_{\subseteq}(C_a, C_b)) )</td>
<td>( KB_{lp} \cup i \text{ memberOf } C_a \models i \text{ memberOf } C_b )</td>
</tr>
<tr>
<td>( \tau_{lp}(KB \models \mu_{\supseteq}(C_a, C_b)) )</td>
<td>( KB_{lp} \cup i \text{ memberOf } C_b \models i \text{ memberOf } C_a )</td>
</tr>
<tr>
<td>( \tau_{lp}(KB \models \mu_{\equiv}(C_a, C_b)) )</td>
<td>( \tau_{lp}(KB \models \mu_{\subseteq}(C_a, C_b)) + \tau_{lp}(KB \models \mu_{\supseteq}(C_a, C_b)) )</td>
</tr>
</tbody>
</table>

Table 5.4: Transformation of matching conditions to WSML-Flight formulas.

We explain the realisation of the entailment \( KB \models \mu(S_r, S_p) \) in the WSML-Flight case exemplarily for the intersection matching condition. Here, we add to the knowledge base \( KB_{lp} \) the fact that some new individual \( i \) is a member of both \( S_r \) and \( S_p \) and check the resulting knowledge for consistency (satisfiability) using the WSML-Flight Reasoner from D1.9. If this check is positive then the matching result between the requester and the provider is positive – otherwise negative. Intuitively, we add, to all the knowledge in the service descriptions and domain ontologies, the fact that there is some individual which conforms to both descriptions \( R_{S_r} \) and \( R_{S_p} \). This is not a problem if these two descriptions are compatible. However, if they are incompatible then this will result in a contradiction in the knowledge base, determined by the reasoner.

**Example of Matchmaking in WSML-Flight**

Here, we give an example of matchmaking applied to a scenario in the travelling domain. In the scenario, consider a service requester who wants to travel from Germany to the UK. We consider two service providers: provider A offers tickets for flights or within Europe; provider B offers offers train tickets in USA. The semantic service descriptions of all the parties are described as follows:

\[
S_r \models Ticket \quad from \quad GermanCity \quad to \quad UKCity \quad vehicle \quad Vehicle
\]

\[
S_{pa} \models Ticket \quad from \quad EUCity \quad to \quad EUCity \quad vehicle \quad Airplane
\]

\[
S_{pb} \models Ticket \quad from \quad USCity \quad to \quad USCity \quad vehicle \quad Train
\]

We list the mappings of these descriptions to WSML-Flight in form of conceptual syntax expressions. (Notice, however, that the instantiation rules cannot be expressed in conceptual syntax.)

\[
\tau_{lp}(R_{S_r}) = \{ \text{concept } S_r \text{ subConceptOf } Ticket \\
\text{from impliesType } EUCity \\
\text{to impliesType } EUCity \\
\text{vehicle impliesType Airplane} \\
x[from hasValue } t_1, to hasValue } t_2, \text{vehicle hasValue } t_3] \models x \text{ memberOf } S_r. \}
\]
\( \tau_{LP}(R_{S_{PA}}) = \{ \text{concept } S_{PA} \text{ subConceptOf FlightTicket} \) \\
\text{from impliesType GermanCity} \\
\text{to impliesType UKCity} \\
\text{vehicle impliesType Vehicle} \\
x [\text{from hasValue } t_1, \text{to hasValue } t_2, \text{vehicle hasValue } t_3] \leftarrow x \text{ memberOf } S_{PA}. \} \)

\( \tau_{LP}(R_{S_{PB}}) = \{ \text{concept } S_{PB} \text{ subConceptOf Ticket} \) \\
\text{from impliesType USCity} \\
\text{to impliesType USCity} \\
\text{vehicle impliesType Train} \\
x [\text{from hasValue } t_1, \text{to hasValue } t_2, \text{vehicle hasValue } t_3] \leftarrow x \text{ memberOf } S_{PB}. \} \)

\[ O_{travel} = \{ \text{concept } Ticket \) \\
\text{from impliesType (1 1) City} \\
\text{to impliesType (1 1) City} \\
\text{vehicle impliesType (1 1) Vehicle} \\
\text{concept FlightTicket subConceptOf Ticket} \\
\text{concept EU City subConceptOf City} \\
\text{concept USCity subConceptOf City} \\
\text{axiom disj_EU_US definedBy} \\
\text{!– } x \text{ memberOf EU City and } x \text{ memberOf USCity.} \\
\text{concept GermanCity subConceptOf EU City} \\
\text{concept UKCity subConceptOf EU City} \\
\text{concept Airplane subConceptOf Vehicle} \\
\text{concept Train subConceptOf Vehicle} \\
\text{axiom disj_Airplane_Train definedBy} \\
\text{!– } x \text{ memberOf Airplane and } x \text{ memberOf Train. } \} \]

If we evaluate the matching conditions \( \mu_{\cap}(S_r, S_{PA}) \) and \( \mu_{\cap}(S_r, S_{PB}) \) according to the translation defined by \( \tau_{LP} \), we get that

\[ \tau_{LP}(R_{S_r}) \cup \tau_{LP}(R_{S_{PA}}) \cup O_{travel} \cup t \text{ memberOf } S_r \text{ and } t \text{ memberOf } S_{PA} \text{ sat.} \]

is true, but

\[ \tau_{LP}(R_{S_r}) \cup \tau_{LP}(R_{S_{PB}}) \cup O_{travel} \cup t \text{ memberOf } S_r \text{ and } t \text{ memberOf } S_{PB} \text{ sat.} \]

is false. This reflects that provider A matches the request while provider B does not, which is exactly what we would intuitively expect. The requester asks for a ticket from Germany to the UK, which both lie in Europe, and does not constrain on the vehicle, such that Airplane is compatible. Contrarily, the requested cities are not compatible with the US location offered by provider B.

### 5.3.3 Matching Service Descriptions in WSML-DL

In this section we map the general formalisation of service descriptions to the WSML-DL ontology language and show how matchmaking can be realised by applying a DL reasoner, such as the one that will be delivered with D1.10.

The syntax and semantics of WSML-DL is, at this point, not complete defined, however, it will most likely coincide with the descriptions logic \( \mathcal{SHIQ} \) [6]. We will use a description logic formal notation in this section as a syntax for WSML-DL expressions.
Description Logics (DL) [6] are fragments of first-order predicate logic and inherit their model-theoretic semantics. Hence, DL formulas are interpreted under open-world assumption. The formulas that make up a DL ontology (or knowledge base) are assertion or inclusion axioms – see [6] for details.

We define the mapping $\tau_{DL}$ for transforming concept and property restrictions, respectively, into WSML-DL formulas in the form of DL axioms in the following table.

<table>
<thead>
<tr>
<th>Restriction</th>
<th>WSML-DL formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{DL}(\rho(C_a, C_b))$</td>
<td>$C_a \sqsubseteq C_b$</td>
</tr>
<tr>
<td>$\tau_{DL}(\rho(C, p, R))$</td>
<td>$C \sqsubseteq \forall \ p. R$</td>
</tr>
</tbody>
</table>

Table 5.5: Transformation of restrictions to WSML-DL formulas.

The transformed sets $\tau_{DL}(R_{S_r})$ and $\tau_{DL}(R_{S_p})$, i.e. application of $\tau_{DL}$ to all restrictions in the sets, are WSML-DL ontologies and build up the knowledge base $KB_{DL}$ together with WSML-DL domain ontologies $O$ as follows

$$KB_{DL} := \tau_{DL}(R_{S_r}) \cup \tau_{DL}(R_{S_p}) \cup \bigcup_{O_i \in O} O_i .$$

We extend the mapping $\tau_{DL}$ to also capture the abstract entailment $KB_{DL} \models \mu(C_a, C_b)$ of matching conditions, transforming it into an application of WSML-DL reasoning services, in the following table.

<table>
<thead>
<tr>
<th>Matching Condition</th>
<th>WSML-DL formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{DL}(KB \models \mu(C_a, C_b))$</td>
<td>$C_a \sqcap C_b$ sat. w.r.t. $KB_{DL}$</td>
</tr>
<tr>
<td>$\tau_{DL}(KB \sqsubseteq C_a, C_b)$</td>
<td>$KB_{DL} \models C_a \sqsubseteq C_b$</td>
</tr>
<tr>
<td>$\tau_{DL}(KB \sqsubseteq C_a, C_b)$</td>
<td>$KB_{DL} \models C_a \sqsubseteq C_b$</td>
</tr>
<tr>
<td>$\tau_{DL}(KB \sqsubseteq C_a, C_b)$</td>
<td>$KB_{DL} \models C_a \sqsubseteq C_b$</td>
</tr>
</tbody>
</table>

Table 5.6: Transformation of matching conditions to WSML-DL formulas.

Similar to the realisation of intersection matching with WSML-Flight, here, in the WSML-DL case, satisfiability of the conjunction of the concepts $S_r$ and $S_p$ ensure that they have a common instance in some model of $KB_{DL}$. Therefore, the positive evaluation of the intersection matching condition also here reflects that the requester and the provider issued compatible service descriptions.

**Example of Matchmaking in WSML-DL**

As before, we also give an example of matchmaking in a travelling scenario for WSML-DL. In the scenario, consider a service requester who wants to travel from Germany to the UK preferably in a sufficiently large and convenient Airplane. We consider three service providers: provider A offers tickets for journeys of all kinds within Europe; provider B offers train tickets in USA; provider C has specialised to cheap flights between England and Germany carried out in a small airplane. The semantic service descriptions of all the parties are described as follows:

---

25
We list the mappings of these descriptions to WSML-DL in form of $SHIQ$ inclusion axioms, and also outline an ontology $O_{travel}$ containing some background knowledge from the travelling domain:

$$
\tau_{DL}(R_{S_r}) = \{ S_r \equiv Ticket \sqcap \\
\forall \text{ from.\,GermanCity} \sqcap \\
\forall \text{ to.\,UKCity} \sqcap \\
\forall \text{ vehicle.\,(Airplane} \sqcap \forall \text{ seats.} \geq 50) \sqcap \}
$$

$$
\tau_{DL}(R_{S_pA}) = \{ S_{PA} \equiv Ticket \sqcap \\
\forall \text{ from.\,EUCity} \sqcap \\
\forall \text{ to.\,EUCity} \sqcap \\
\forall \text{ vehicle.\,Vehicle} \}
$$

$$
\tau_{DL}(R_{S_pb}) = \{ S_{PB} \equiv Ticket \sqcap \\
\forall \text{ from.\,USCity} \sqcap \\
\forall \text{ to.\,USCity} \sqcap \\
\forall \text{ vehicle.\,Train} \}
$$

$$
O_{travel} = \{ Ticket \sqsubseteq \exists \text{ from.}\top \sqcap \exists \text{ to.}\top \sqcap \exists \text{ vehicle.}\top, \\
EUCity \sqsubseteq \text{City}, \; USCity \sqsubseteq \text{City}, \; UKCity \sqsubseteq \text{EUCity}, \\
\text{EnglishCity} \sqsubseteq \text{UKCity}, \; \text{GermanCity} \sqsubseteq \text{EUCity}, \\
EUCity \sqcap \text{USCity} \sqsubseteq \bot, \; \text{UKCity} \sqcap \text{GermanCity} \sqsubseteq \bot, \\
\text{Airplane} \sqsubseteq \text{Vehicle}, \; \text{Train} \sqsubseteq \text{Vehicle}, \; \text{Airplane} \sqcap \text{Train} \sqsubseteq \bot, \\
\text{Airplane} \sqsubseteq \exists \text{ seats.}\,\text{integer} \}
$$

If we evaluate the matching conditions $\mu_\cap(S_r, S_{PA})$, $\mu_\cap(S_r, S_{PB})$ and $\mu_\cap(S_r, S_{PC})$ according to the translation defined by $\tau_{DL}$, we get that

$$
S_r \sqcap S_{PA} \text{ sat. w.r.t. } \tau_{DL}(R_{S_r}) \cup \tau_{DL}(R_{S_pA}) \cup O_{travel}
$$
is true, but

$$
S_r \sqcap S_{PB} \text{ sat. w.r.t. } \tau_{DL}(R_{S_r}) \cup \tau_{DL}(R_{S_pb}) \cup O_{travel}
$$
and

$$
S_r \sqcap S_{PC} \text{ sat. w.r.t. } \tau_{DL}(R_{S_r}) \cup \tau_{DL}(R_{S_pC}) \cup O_{travel}
$$

$^1$Notice that here we use nested constructs for temporary concepts like $\text{Airplane}_1$. 
are false. This reflects that provider A matches the request while providers B and C do not, which is what we would intuitively expect. The requester’s constraint on the airplane being sufficiently large rules out provider C who supports small airplanes only. Provider B is ruled out because it does not offer flight tickets at all.

### 5.3.4 Discussion of the Matchmaking Approach

We finally discuss some characteristics of our matchmaking approach which we think are important for its use within the DIP environment.

**Carefully Modelling Domain Knowledge**

From the examples it can be seen that the chosen matchmaking approach is reliant on forms of “negative knowledge”, like disjointness constraints or negation, which one would expect to be contained in consciously engineered domain ontologies. If such constraint information is not present, the matchmaking algorithm is likely to produce false positive matches. For example, if the request for a flight ticket modelled in terms of a travelling ontology is matched against service offers in some unrelated domain, say financial services modelled in terms of different ontologies, then there will probably be no constraint in the knowledge base stating that e.g. tickets are incompatible with mortgages. However, one way to avoid such potential false positives is to employ in the discovery process a precedent non-semantic categorisation step, as mentioned in Section 3.3. This could, for example, be based on keyword matching or even on taxonomic classification of the involved top-level concepts like Ticket or Loan. In this way, one could avoid to enter the semantic matchmaking process for comparing tickets with mortgages, if the two categories are not taxonomically connected. A service request would be associated to the “right” repository of service offer descriptions beforehand, each such repository being a controlled environment for which well-defined and carefully modelled ontologies can be maintained, restricted in scope to some particular domain of value.

This is also related to the combination of discovery with mediation mentioned in Section 3.4, which will be investigated in the scope of work on Deliverable D5.7 in WP5.

**Restricting Datatype Ranges**

Currently, the mapping of property restrictions \( \rho(C, p, R) \) to WSML-Flight does not support datatype ranges. The problem is that the approach in the WSML-Flight case relies on an instantiation of “dummy” individuals which are then checked to meet the constraints imposed by both the service offer and the service request description. This works for instantiating individuals but not for instantiating data values, since it is not clear which concrete value to take as an instance when an interval of values is specified.

A characteristic of datalog-style languages with closed-world semantics, as WSML-Flight is, is that they are more suitable to operate on knowledge bases which store ground facts and to querying for individuals. A fact like “the value of property \( p \) for concept \( C \) is less than 5”, however, is non-ground and cannot be an answer to a query. For such languages, matching approaches based on retrieval and querying seem to be more suitable.

However, we argue that a retrieval-based approach, querying for instances in a knowledge base does not capture the symmetry we have between requester and provider descriptions in our approach. Namely, both requesters and providers can encode variability in their semantic service descriptions allowing for multiple intended concrete services, as described
in a former section. Contrarily, a retrieval approach would asymmetrically restrict one side to list all their concrete services in form of ground facts to be queried by the other side.

The mapping to the WSML-DL variant does not have this problem with datatype ranges, since in description logics this can be handled using concrete domain extensions. Our conclusion is, that the open-world paradigm that WSML-DL employs does better fit the matchmaking approach described here, which is due to the modelling of service capabilities in form of concepts being templates for service configurations.

**Ranking Service Offers**

We have formalised different matching conditions $\mu \cap$, $\mu \subseteq$, $\mu \supseteq$ and $\mu \equiv$, which are in some approaches used to establish a ranking of relevant service offers. An implementation of the discovery component could use them to realise the different degrees of matching such a ranking is based on. This notion of ranking gives information on the options an agent has in the subsequent service definition phase – a service with a higher rank provides more options to select the concrete service to be carried out than one with a lower rank does (see [14] for details).

However, we argue that a ranking of service offers in terms of their relevance for a request strongly depends on the concrete scenario in which discovery is applied. In concrete use cases, an agent requesting a particular functionality typically decides which service to use by taking into account preference information that is not included in semantic descriptions of services or the domain models. In commercial or financial scenarios as we have them in WP10, for example, it is often desired to have a ranking of services according to their price or other such parameters, which are negotiated during the service definition phase. Therefore, we see a practical ranking for concrete use cases outside a generic discovery component, residing in the agent’s behaviour.

**Delivering Multiple Objects**

The use of the notion of “object delivered by the service” brings up the question of how to handle a service that delivers multiple objects, which has e.g. been picked up in [16]. An example would be a service that offers a skiing holiday including travelling by train, the hotel to stay at and some ski ticket. This brings in a compositional aspect, which is in general not subject of this deliverable but addressed e.g. in D4.12. The matchmaking approach described here does not directly support “multiple objects” in this sense, as in [16], but could probably be applied repeatedly within some form of composition strategy.

From a description point of view, the aspect of multiple objects could be covered in two ways. One would be to split up a service in different actual Web Services each of which has its own capability description reflecting a single object. Another way would be to choose an appropriate modelling that captures the bundle of objects, making this bundle the “object delivered by the service”.
6 DISCOVERY IN DIP USE CASES

In this chapter we link to the use case work packages, where the discovery component is applied in concrete scenarios. Reflecting the current state of cooperation with the use case partners, we refer to the first prototype of the telecommunication use case from WP8 and to the “application 1” eBanking use case from WP10. The “change of circumstances” scenario in WP9 turned out to not require discovery. In future work, the GIS-related use cases from WP10, as well as the “application 2” in WP10 are also to be investigated in the context of discovery.

6.1 Telecommunications

The scenario around the second prototype of the telecommunications use case in WP8 is concerned with the management of a contract catalogue of telecommunication products, and will be described in detail in D8.5.

Within this scenario, a contract catalogue is a collection of Information and Communication Technology (ICT) products and services that are offered to a customer by BT. The different products are sourced from many internal and external suppliers, which BT brings together and offers to the customer as a single catalogue.

The specific problems associated with managing the contract catalogue are:
- arranging and managing a set of products from many sources to present a consistent view to the customer
- interfacing with the third party suppliers for order querying, processing and fulfilment
- managing the work flow of complex products
- producing product ‘bundles’ that may have constraints or dependencies between them.
- identifying dependencies that arise due to existing products and services (i.e. the inventory).
- representing dependencies as rules and propagating them between the various levels of service offerings.

Currently BT is moving towards using Web Services to enable a more consistent interface between its self and the suppliers. The case study aims to show that using Semantic Web Services can add more benefit by allowing the product capabilities and rules to be modelled and exposed as part of the semantic description of the Web Service.

The application for discovery in this scenario is in the creation of product bundles. Product bundles bring together complementary products and services to enable the customer to order a complete solution for a particular need. An example of this would be a ”Trader Workstation”. This bundle would offer all the ICT components needed for a trader to carry out their job. This would include such things as:
- Share Price Feed
- Network connection
- Phone line
- Desktop PC
- O/S Software
- Trading Software
A BT Product Manager would be responsible for designing this bundle and currently it is a manually intensive process. Each component of the bundle will have specific rules or constraints associated with them and it is up to the product manager to make sure that the components are compatible. For example the 'Share Price Feed' is a service to provide live share price information. A Requirement of this feed is that it needs at least 50Kbit/sec of bandwidth over the Internet to run. This means that the 'Network Connection' must provide at least this.

If we use a domain ontology to represent generic products and their particular constraints and requirements, it would allow a product manager to dynamically build a set of goals to represent the restrictions on the products they require in the bundle (via an interface that hides the actual WSML representation). If each provider uses SWS to describe the capabilities of their products on offer, it would enable the discovery engine to match suitable products to the requirements.

As an example, consider the following two requests that a product manager issues when configuring a product bundle.

\[
S_{r_1} \simeq \text{NetworkConnection} \quad \frac{\text{bandWidth}}{} \geq 50 \text{ kbit/sec} \\
S_{r_2} \simeq \text{Desktop}
\]

They ask for a network connection and a desktop computer as parts of the bundle. The restriction on the network bandwidth stems from the constraint for the share price feed feature mentioned before. It could be part of a telecommunications domain ontology, as depicted in Figure 6.1.

Figure 6.1: Information within a telecommunications domain ontology.

Several external providers offer the potential modules for composing the bundle. As the product manager, they refer to the domain ontology depicted in Figure 6.1, describing their offers as follows.

\[
S_{pA} \simeq \text{ADSLConnection} \quad \frac{\text{bandWidth}}{} \rightarrow 1024 \text{ kbit/sec} \\
S_{pB} \simeq \text{DialupConnection} \quad \frac{\text{bandWidth}}{} \rightarrow 10 \text{ kbit/sec} \\
S_{pC} \simeq \text{Workstation} \quad \frac{\text{memory}}{	ext{CPU}} \rightarrow \geq 1 \text{ GB} \geq 2 \text{ GHz} \\
S_{pD} \simeq \text{Desktop, Laptop} \quad \frac{\text{memory}}{	ext{CPU}} \rightarrow \leq 2 \text{ GB} \leq 4 \text{ GHz}
\]
As can be easily verified from the parameter restrictions, the discovery engine would identify provider A as relevant for request 1, and provider D as relevant for request 2. The constraint on the network connection ensures that the bundle is equipped with sufficient bandwidth (not supplied by the dialup connections of provider B), while asking for a lean desktop solution rules out the high-performance workstations (offered by provider C).

6.2 eBanking

The scenario around application 1 of the eBanking use case is concerned with the comparison of mortgage services in the financial field. It is in detail described in D10.5 and other WP10 documentation.

In this scenario, a mortgage comparator application acts as a broker to financial institutions offering mortgage services via the web. The application provides means to compare several currently offered mortgage services and gives the user a summarised view on them. To reflect the current market situations, the application needs to get aware of new mortgage offerings that enter the market. Here service discovery comes into play: the application identifies those mortgage offers, accessible through Web Services, which are relevant for the current user request. The mortgage services involved are described by financial parameters of the mortgage loan objects, like the loan interest rate, the total mortgage amount, the real estate the mortgage is provided for, etc.

We give a (modified) example, taken from the mortgage comparison use case described in D10.5, to illustrate discovery in eBanking. The following shows the semantic service descriptions of a requester and of different financial providers A and B, in the style introduced in Section 5.

\[
S_r \simeq \frac{\text{Mortgage interestRate}}{\text{totalAmount}} \leq 4\% \quad \frac{\text{totalAmount}}{\text{realEstate}} \leq 200,000.00\€ \quad \text{Apartment}
\]

\[
S_{PA} \simeq \frac{\text{Loan interestRate}}{\text{totalAmount}} \geq 3\% \quad \frac{\text{totalAmount}}{\text{realEstate}} \geq 30,000.00\€
\]

\[
S_{PB} \simeq \frac{\text{Mortgage interestRate}}{\text{totalAmount}} \geq 4.5\% \quad \frac{\text{totalAmount}}{\text{realEstate}} \geq 150,000.00\€ \quad \text{IndustrialFacility}
\]

The financial domain ontologies the requester and provider parties commonly refer to define notions like Loan, interestRate, Mortgage, etc. Figure 6.2 shows an excerpt of such ontologies.

The domain ontologies define taxonomies relevant for the financial items offered by service providers. Besides subsumption, they also specify disjointness relations between siblings in the taxonomies. This is important for the matchmaking mechanism in the discovery process, since otherwise it would not recognise that e.g. Credit and Savings are incompatible classes of objects.

Together with the taxonomic information depicted in Figure 6.2, matchmaking applied to the descriptions given above yields the result that provider A matches the request while provider B does not. The offerings of provider A lie within the scope of the requested parameters and therefore the requester can potentially find an agreement with this provider in a subsequent service definition phase. On the other hand, provider B does not fulfil the
requested requirements, for two reasons: the requested maximum interest rate is exceeded, and the specialisation to mortgages on industrial facilities does not fit the requirement for a mortgage on an apartment.
A An ASM Specification of the Discovery Framework

ASM models can help to provide explicit, exact and formal specifications with an accurate meaning of all underlying terms, needed to produce a consistent view of the general SWS usage process. Furthermore, the ASM method allows us to isolate the hard part of a system, e.g. communication and concurrency issues and thus to concentrate on the essential parts for refinement, targeted at bridging controversial approaches, like different distribution strategies or different semantic matchmaking approaches, through explicitly showing their differences by deriving them as different refinements of the same abstractions.

A.1 Formalization of SWS discovery framework

We see the discovery service as an interface, which is defined by the following methods:

- **ReceiveGoal** for receiving goal queries (elements of a set $InGoalMssg$) from clients,
- **SendSetOfWS** for sending sets of found Web Services (elements of a set $OutWSMssg$) back to clients,
- **ProcessGoal** to handle $ReceivedGoals$ (elements of a set $GoalObj$ of internal representations of received goals, say as goal objects), typically by sending to relevant discovery service providers goal queries, which are needed to service the currently handled goal request $currGoalObj$,
- **SendGoal** for sending outgoing goal queries (elements of a set $OutGoalMssg$) to providers, which may again be discovery service providers,
- **ReceiveSetOfWS** for receiving incoming sets of found Web Services (elements of a set $InWSMssg$) from discovery service providers.

We define discovery service provider as an ASM, which at each moment chooses one of its submachines (non-deterministically) for execution (where we abstain from representing here the selection of the parameters involved in such submachine calls):

$$\text{DiscoveryServiceProvider} = \text{choose } M \in \{\text{ReceiveGoal, SendSetOfWS}\} \cup \{\text{ProcessGoal, SendGoal, ReceiveSetOfWS}\}$$

We formulate a “stateless” model for the communication between clients and discovery service providers, which assumes that the relevant state information for every received or sent message is contained in the message. The notion of state is restricted to the detection of loops, which can be caused by other discovery service providers sending a goal that is already in the processing by the receiving provider. Therefore, a goal must be uniquely identifiable in the global context.

---

1Since instances of the abstract machines DiscoveryServiceProvider we are going to define here can be distributed over multiple discovery locations (see Sect. subsection:distributing), such a client can also be another (Virtual) Discovery Service Provider DSP asking for servicing a goal query of a goal received by DSP.

2We deliberately keep the underlying message passing system abstract, so that the scheme we are going to develop for Discovery Service Providers can be instantiated in such a way that also ProcessGoal itself can be a provider of a discovery location and thus service a goal query ‘internally’.
A.1.1 Abstract Message Passing

For receiving and sending request and answer messages we abstract from the particularities of a concrete message passing system. This means that we use the following behavioral communication interfaces (to be imported predicates and machines, specified and implemented elsewhere) for mail boxes of incoming and outgoing messages.

- a predicate `ReceivedGoal`, used by `ReceiveGoal` and expressing that an incoming goal query (which is passed as argument to the predicate) has been received from some client (which we suppose to be encoded into the message),
- a predicate `ReceivedSetOfWS` used by `ReceiveSetOfWS` and expressing that a message containing a set of web services (to a previously sent goal query, which we suppose to be retrievable from the message) has been received,
- a machine `Send`, used a) by `SendSetOfWS` for sending out messages containing sets of web services to goal queries back to the clients where the goal queries originated, b) by `SendGoal` for sending out goal queries to providers. In both cases we assume the addresses to be known or encoded into the message.

For reasons of modularity we separate the internal preparation of outgoing answer or request messages in `ProcessGoal` from the machine `Send` which does the actual sending and relates to the communication medium we want to keep abstract. We therefore use in addition to `ReceivedGoal` and `ReceivedSetOfWS` the following two abstract predicates for mail boxes of outgoing mail:

- `SentSetOfWSToRequestor` expressing that an outgoing message containing a set of found web services (elaborated from a `ProcessGoal` internal representation of an answer) has been sent to `Send`, to be actually passed to the external message passing system,
- `SentGoalToProvider` expressing that an outgoing goal query (corresponding to an internal representation of a goal) has been sent to `Send` to be actually passed to the external message passing system.

A.1.2 The Send/Receive submachines

The interaction between a client (initial requestor or some discovery service provider) and a discovery service provider, which is triggered by the arrival of a client’s goal query via the message passing system, is characterized by creating at the DSP a goal object (a goal ID, say element `g` of a set `GoalObj` of currently alive goal objects), which is appropriately initialized by recording in an internal representation the relevant data, which are encoded in the received goal query. This includes decorating that object by an appropriate `status`, say `status(g) := started` for a new goal or `status(g) := loopDetected` for a goal already in processing at this discovery location, to signal to `ProcessGoal` its readiness for being processed, and by other useful information. That is, for the same initial goal several `inGoalMsg` can arrive at a discovery location causing the creation of a new element `g` of a set `GoalObj` for each of them. However, only one `g` for the same initial goal will get the status `started` and will be really processed while all the others will get the status `loopDetected` causing `ProcessGoal` to create an answer containing an empty set of web services.

This requirement for the machine `ReceiveGoal` is captured by the following definition, which is parameterized by the incoming request message `inGoalMsg` (supposed to belong to the set `InGoalMssg` of legal incoming goal queries) and by the set `GoalObj` of
current goal objects of the DiscoveryServiceProvider. For simplicity of exposition we assume a preemptive ReceivedGoal predicate\(^3\).

\[
\text{RECEIVEGOAL}(\text{inGoalMsg}, \text{GoalObj}) = \\
\quad \text{if } \text{ReceivedGoal}(\text{inGoalMsg}) \text{ then} \\
\quad \quad \text{let } g = \text{New}(\text{GoalObj})^4 \text{ in} \\
\quad \quad \quad \text{INITIALIZE}(g, \text{inGoalMsg}) \\
\quad \quad \quad \text{INITIALIZE}(\text{SetOfWS}(g)) \\
\quad \quad \quad \text{if } \text{NewGoal}(g, \text{GoalObj}) \text{ then} \\
\quad \quad \quad \quad \text{status}(g) := \text{started} \\
\quad \quad \quad \text{else} \\
\quad \quad \quad \quad \text{status}(g) := \text{loopDetected} \\
\]

\[
\text{INITIALIZE}(\text{SetOfWS}(g)) = \\
\quad \text{SetOfWS}(g) := \emptyset
\]

The interaction between a discovery service provider and a client, which consists in sending back a message containing a set of found web services to a previous goal query of the client, is characterized by the underlying goal object having reached through further processing of PROCESSGOAL a status where a call to SENDSETOFWS with corresponding parameter outSetOfWSMsg has been internally prepared by PROCESSGOAL — say by setting an answer-mailbox predicate SentSetofWSToRequestor for this argument to true. Thus one can specify SENDSETOFWS, and symmetrically SENDGOAL with a request-mailbox predicate SentGoalToProvider, as follows: \(^5\)

\[
\text{SENDSETOFWS}(\text{outSetOfWSMsg}, \text{SentSetOfWSToRequestor}) = \\
\quad \text{if } \text{SentSetOfWSToRequestor}(\text{outSetOfWSMsg}) \text{ then} \\
\quad \quad \text{SEND}(\text{outSetOfWSMsg})
\]

\[
\text{SENDGOAL}(\text{outGoalMsg}, \text{SentGoalToProvider}) = \\
\quad \text{if } \text{SentGoalToProvider}(\text{outGoalMsg}) \text{ then} \\
\quad \quad \text{SEND}(\text{outGoalMsg})
\]

For the definition of RECEIVESETOFWS we use as parameter the SetOfWS function which provides for every goal \(g\), which may have triggered sending some goal queries to other discovery service providers, the \(\text{SetOfWS}(g)\), where to insert (the internal representation of) each \(\text{setOfWS}\) contained in the incoming message. \(^6\)

\[
\text{RECEIVESETOFWS}(\text{inSetOfWSMsg}, \text{SetOfWS})^7 = \\
\quad \text{if } \text{ReceivedSetOfWS}(\text{inSetOfWSMsg}) \text{ then} \\
\quad \quad \text{insert } \text{SetOfWS}(\text{inSetOfWSMsg}) \text{ into } \text{SetOfWS}(\text{goal}(\text{inSetOfWSMsg}))
\]

\(^3\)Otherwise a DELETE(\text{inGoalMsg}) has to be added with the effect that the execution of RECEIVEGOAL(\text{inGoalMsg}, \text{GoalObj}) switches ReceivedGoal(\text{inGoalMsg}) from true to false.

\(^4\)New is assumed to provide at each call a sufficiently fresh element in the indicated domain.

\(^5\)For the sake of generality, we assume the destinators of messages to be encoded into the message.

\(^6\)The function \(\text{goal}(\text{inSetOfWSMsg})\) is defined below to denote the goal \(g\) to which the \(\text{inSetOfWSMsg}\) is received now.

\(^7\)Without loss of generality we assume this machine to be preemptive (i.e. ReceivedSetOfWS(\text{inSetOfWSMsg}) gets false by firing RECEIVESETOFWS for \text{inSetOfWSMsg}).
Behavioral interface types. Through the definitions below, we will link ReceiveGoal and SendSetOfWS by the status function value for a currGoalObj. This realizes that the considered communication interface is of the “provided behavioral interface” type, discussed in [8], where the ReceiveGoal action corresponds to receive an incoming request, through which a new goalObj is created (initStatus(PROCESSGOAL)), and occurs before the corresponding SendSetOfWS action, which happens after the outgoing answer message in question has been SentSetOfWSToRequestor, namely when goalObj was reaching the status deliver. The pair of machines SendGOAL and ReceiveSetOfWS in PROCESSGOAL (more precisely in the submachine ITERATESUBREQPROCESS defined below) realizes the symmetric “required behavioral interface” communication interface type, where the Send actions in SendGOAL correspond to outgoing goal queries and thus occur before the corresponding ReceiveSetOfWS actions of the incoming sets of found web services to those goal queries.

A.2 The PROCESSGOAL Submachine

In this section we define the signature and the transition rules of the Abstract State Machine PROCESSGOAL for the processing kernel of a service discovery provider. The signature definition really provides a schema, which is to be instantiated for each particular PROCESSGOAL kernel of a concrete Service Discovery Provider, namely by giving concrete definitions for the abstract functions we are going to introduce (see their listing in the documentation section below).

In our definition we want to abstract from the scheduling mechanism which calls PROCESSGOAL for a particular current goal object currGoalObj. We therefore describe the machine as parameterized by such a currGoalObj ∈ GoalObj, which plays the role of a global instance variable. The definition is given in terms of control state ASMs in Fig. A.1, using the standard graphical representation of finite automata or flowcharts as graphs with circles (for the internal states, here to be interpreted as current value of status(currGoalObj)), rhombs (for test predicates) and rectangles (for actions).

The definition in Fig. A.1 expresses that each PROCESSGOAL call for a started goal object currGoalObj triggers to FEEDSENDGOAL with a goal query to be sent out for every relevant discovery location l of the current currGoalObj, namely by setting SentGoalToProvider(outGoal2Mssg(l)) to true. Here outGoal2Mssg(l) transforms the outgoing goal query into the format for an outgoing goal query message, which has to be an element of OutGoalMssg. Since those immediate goal queries, elements of a set ParGoalQuery (currGoalObj), are assumed to be processable by other discovery service providers independently of each other, FEEDSENDGOAL elaborates simultaneously for each l an outGoalMssg(l). Simultaneously, PROCESSGOAL updates the status of currGoalObj to status(currGoalObj) := waitingForAnswers, where currGoalObj remains until AllAnswersReceived or BreakCondition becomes true.

PROCESSGOAL call for a loopDetected updates the status of currGoalObj to status(currGoalObj) := compileAnswer.

As long as during waitingForAnswers, AllAnswersReceived(currGoalObj) is not yet true and BreakCondition(currGoalObj) is false, RECEIVESETOFWS inserts for every ReceivedSetOfWS(inSetOfWSMssg) the retrieved internal setOfWS(inSetOfWSMssg) representation into SetOfWS(currGoalObj) of the currently processed goal currGoalObj, which is supposed to be retrievable as goal of the incoming answer message.
Once ProcessGoal finds AllAnswersReceived(currGoalObj) or the BreakCondition (currGoalObj) becomes true currGoalObj assumes status value compileAnswer.

Once, ProcessGoal finds status(currGoalObj) has value compileAnswer it compiles from currGoalObj (which allows to access SetOfWS(currGoalObj)) an answer, say outSetOfWS(currGoalObj), and transforms the internal answer information a into an element of OutWSMmsg using an abstract function outSetOfWS2Mssg(a). We guard this answer compilation by a previous check whether SetOfWSToBeSent for the currGoalObj (including the SetOfWS(currGoalObj)) evaluates to true.

For the sake of illustration we also provide here the textual definition of the machine defined in Fig A.1.

\[
\text{ProcessGoal}(\text{currGoalObj}) = \\
\text{if status(currGoalObj) = started then} \\
\text{FeedSendGoal with ParGoalQuery(currGoalObj)} \\
\text{status(currGoalObj) := waitingForAnswers} \\
\text{if status(currGoalObj) = loopDetected then} \\
\text{status(currGoalObj) := compileAnswer} \\
\text{if status(currGoalObj) = waitingForAnswers then} \\
\text{if BreakCondition(currGoalObj) then} \\
\text{GenerateException(currGoalObj)} \\
\text{status(currGoalObj) := compileAnswer} \\
\text{else} \\
\text{if AllAnswersReceived(currReqObj) then} \\
\text{status(currGoalObj) := compileAnswer} \\
\text{if status(currGoalObj) = compileAnswer then} \\
\text{CompileOutSetOfWSMsg from currGoalObj}
\]
\[ \text{status(currGoalObj)} := \text{deliver} \]

where

\[
\text{FeedSendGoal with ParGoalQuery(currGoalObj)} =
\]
\[
\text{forall } s \in \text{ParGoalQuery(currGoalObj)}
\]
\[
\text{SentGoalToProvider(outGoal2Msg(s))} := \text{true}
\]

\[
\text{CompileOutSetOfWSMsg from currGoalObj} =
\]
\[
\text{SentSetOfWSRequestor(outSetOfWS2Msg(outSetOfWS(currGoalObj)))} := \text{true}
\]

Sets of WS.

\[
\text{AllAnswersReceived} =
\]
\[
\text{for each req } \in \text{ParGoalQuery(currGoalObj)}
\]
\[
\text{there is some answ } \in \text{SetOfWS(currGoalObj)}
\]

A.3 DISCOVERYENGINE ASM

Similar to DISCOVERYSERVICEPROVIDER, we see the discovery engine as an interface, which is defined by the following methods:

- **ReceiveGoal** for receiving goal queries (elements of a set \( \text{InGoalMsg} \)) from a client\(^8\),
- **SendSetOfWS** for sending sets of found Web Services (elements of a set \( \text{OutWSMsg} \)) back to the associated DSP,
- **MatchGoal** to handle ReceivedGoals (elements of a set \( \text{GoalObj} \) of internal representations of received goals, say as goal objects), typically by filtering and matching the locally available set of web services to service the currently handled goal request \( \text{currGoalObj} \).

We define a discovery engine as an ASM, which at each moment chooses one of its submachines (non-deterministically) for execution (where we abstain from representing here the selection of the parameters involved in such submachine calls):

\[
\text{DISCOVERYENGINE} =
\]
\[
\text{choose } M \in \{\text{ReceiveGoal, SendSetOfWS}\} \cup \{\text{MatchGoal}\}
\]
\[
M
\]

The machine SendSetOfWS has been reused from DISCOVERYSERVICEPROVIDER. The machine ReceiveGoal from DISCOVERYSERVICEPROVIDER has been slightly modified.

\[
\text{ReceiveGoal}(\text{inGoalMsg, GoalObj}) =
\]
\[
\text{if ReceivedGoal(\text{inGoalMsg}) then}
\]
\[
\text{let } g = \text{New(\text{GoalObj}) in}
\]

\(^8\)Each instance of the abstract machines DISCOVERYENGINE we are going to define here is associated with a DISCOVERYSERVICEPROVIDER, i.e., only the associated Discovery Service Provider DSP asking for servicing a goal query of a goal received by DSP will be served.
$\text{Initialize}(g, \text{inGoalMsg}, \text{inSetOfWS}^9)$
$\text{Initialize}((\text{SetOfWS}(g))$

\[ \text{status}(g) := \text{started} \]

\text{MatchGoal} executes the machines PREFILTERING, SEMANTICMATCHMAKING and QOSMATCHMAKING sequentially reducing stepwise the initial set of web services \text{inSetOfWS} to the final set of web services matching the goal. The final set is sent to the DSP as soon as \text{currGoalObj} is set to \text{status(currGoalObj)} := \text{compileAnswer}.

$\text{MatchGoal}(\text{currGoalObj}) =$

\begin{align*}
\text{if} \ & \text{status(currGoalObj)} = \text{started} \quad \text{then} \\
& \text{PREFILTERING(currGoalObj)} \\
& \text{status(currGoalObj)} := \text{filtered} \\
\text{if} \ & \text{status(currGoalObj)} = \text{filtered} \quad \text{then} \\
& \text{SEMANTICMATCHMAKING(currGoalObj)} \\
& \text{status(currGoalObj)} := \text{matched} \\
\text{if} \ & \text{status(currReqObj)} = \text{matched} \quad \text{then} \\
& \text{QOSMATCHMAKING(currGoalObj)} \\
& \text{status(currGoalObj)} := \text{compileAnswer} \\
\text{if} \ & \text{status(currGoalObj)} = \text{compileAnswer} \quad \text{then} \\
& \text{COMPILOUTSetOfWSMsg from currReqObj} \\
& \text{status(currGoalObj)} := \text{deliver} \\
\text{where} \\
& \text{COMPILOUTSetOfWSMsg from currReqObj} = \\
& \text{SentSetOfWSToRequestor(outSetOfWS2Mssg(outSetOfWS(currReqObj))))} := \text{true}
\end{align*}

The machines PREFILTERING, SEMANTICMATCHMAKING and QOSMATCHMAKING can now be further refined in order to implement different filtering and matchmaking methods or strategies.

The introduced high-level ASM model of distributed semantic discovery process provides a basis for communicating and documenting design ideas and supports an accurate and checkable overall understanding of the controversially discussed topic of frameworks and methods for semantic WS discovery, a part of the Semantic Web Services (SWS) usage process [11]. The presented formalization of the discovery framework demonstrates that the different distribution and semantic matchmaking strategies, depending on the technology used for an implementation of a discovery service, can be derived as different refinements of the same abstractions. We used the concept of the Virtual Provider (VP) as a basis for the formal specification of the distributed semantic discovery framework [5]. Only minor changes on the VP structure were required in order to specify a formal, high-level ASM model of distributed semantic discovery services. Besides the discovery topic, VP has proven to be useful also in the area of mediation and composition, [3], [4], [7].

\[ ^9 \text{inSetOfWS} \] is assumed to contain an initial set of web services to be matched with the goal $g$. 
REFERENCES


