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

Deliverable

WP4: Service Usage
D4.17
P2P & QoS-enabled Service Discovery Specification

Manfred Hauswirth, Fabio Porto, Le-Hung Vu

June 27, 2006
SUMMARY

Semantic Web service discovery is the process of automatically finding Web services that fulfill a certain user goal. Typically, users express a goal by specifying the functionalities Web services should provide in order to achieve it. In this deliverable, we augment the notion of functionality-base Semantic Web service discovery to consider quality of service (QoS) criteria. We extend goals and Web service descriptions to support the specification of QoS parameters and provide an architecture to integrate both QoS and functionality-based service discovery within the context of the WSMX architecture. A particularly relevant aspect of our approach is a new trust and reputation model that prevents Web service providers from cheating on QoS parameters of published Web services. To demonstrate the applicability of QoS-based service discovery to real-world scenarios, we have specified an application scenario in common agreement with the e-Banking use case partner (WP10). Another working example is also being formulated through the negotiation with the WP8 working on the B2B Telecom case study.

This deliverable contributes to the Semantic Web Services Initiative (SWSI) according to the following aspects:

- Ontology languages: We extend the WSMO ontology with QoS semantics.
- Ontological infrastructure and Web service architecture: We extend functionality-based web service discovery with QoS-based discovery.

Regarding the development of exploitable tools, this deliverable presents the specification of the QoS-based service discovery component to be developed in D4.18 and D4.19. The QoS-based discovery is a component integrated within the WSMX architecture and in particular designed in accordance with partners specifying the functionality-based discovery component in D4.8.

Potential readers of this deliverable are:

- WP1 - extensions of the WSMO model to support QoS parameters
- WP2 - requirements for the repository interface for retrieving Web service descriptions and ontologies as input parameters for the discovery process
- WP4 - information concerning changes to WSMO tools to support a QoS extension of WSMO; information regarding QoS-based discovery for supporting dynamic composition; information regarding the integration of functional and QoS-based discovery
- WP5 - requirements for mediating goals with Web service descriptions defined using different ontologies, during semantic matchmaking
- WP6 - information regarding the integration of the discovery component into the WSMX architecture
- WP8 - use case partner defining a B2B Telecom case study with QoS
- WP10 - use case partner defining an e-banking case study with QoS
• Other partners interesting in QoS-based Semantic Web service discovery and its applications

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.
**Abstract (for dissemination)**

Semantic Web service discovery is the process of automatically finding Web services that fulfill a certain user goal. Typically, users express a goal by specifying the functionalities Web Services should provide in order to achieve the goal. In this deliverable, we augment the notion of functionality-based Semantic Web service discovery to consider quality of service (QoS) criteria. We extend goals and Web service descriptions to support the specification of QoS parameters and provide an architecture to integrate both QoS and functionality-based Semantic Web service discovery within the context of the WSMX architecture. A particularly relevant aspect of our approach is a new trust and reputation model that prevents Web service providers from cheating on QoS parameters of published Web services. To demonstrate the applicability of QoS-based service discovery to real-world scenarios, we have specified an application scenario in common agreement with the e-Banking use case partner (WP10). Another working example is also being formulated through the negotiation with the WP8 working on the B2B Telecom case study.

**Keywords**

Semantic Web service, SWS, service discovery, QoS, Goal, API
## Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-05-2005</td>
<td>1</td>
<td>Fabio Porto</td>
<td>v 0.1</td>
</tr>
<tr>
<td>27-05-2005</td>
<td>2</td>
<td>Manfred Hauswirth</td>
<td>v 0.2</td>
</tr>
<tr>
<td>08-06-2005</td>
<td>3</td>
<td>Roman Schmidt</td>
<td>v 0.3</td>
</tr>
<tr>
<td>27-07-2005</td>
<td>4</td>
<td>Fabio Porto</td>
<td>v 0.4</td>
</tr>
<tr>
<td>03-08-2005</td>
<td>5</td>
<td>Le-Hung Vu</td>
<td>v 0.5</td>
</tr>
<tr>
<td>07-10-2005</td>
<td>6</td>
<td>Fabio Porto</td>
<td>v 0.6</td>
</tr>
<tr>
<td>06-12-2005</td>
<td>7</td>
<td>Le-Hung Vu</td>
<td>v 0.7</td>
</tr>
<tr>
<td>12-12-2005</td>
<td>8</td>
<td>Le-Hung Vu, Fabio Porto,</td>
<td>v 0.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manfred Hauswirth</td>
<td></td>
</tr>
<tr>
<td>22-12-2005</td>
<td>10</td>
<td>Le-Hung Vu, Fabio Porto,</td>
<td>v 1.0 - Final version for internal reviewers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manfred Hauswirth</td>
<td></td>
</tr>
<tr>
<td>27-01-2006</td>
<td>11</td>
<td>Le-Hung Vu, Fabio Porto,</td>
<td>v 1.1 - Minor changes in the proposed extension to WSMO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manfred Hauswirth</td>
<td>(section 2.4.2 and section 4.)</td>
</tr>
<tr>
<td>12-06-2006</td>
<td>12</td>
<td>Le-Hung Vu, Fabio Porto,</td>
<td>v 1.2 - First changes towards the comments of the final reviewers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manfred Hauswirth</td>
<td></td>
</tr>
<tr>
<td>21-06-2006</td>
<td>13</td>
<td>Fabio Porto</td>
<td>v 1.3</td>
</tr>
<tr>
<td>26-06-2006</td>
<td>14</td>
<td>Manfred Hauswirth</td>
<td>v 1.4</td>
</tr>
<tr>
<td>30-06-2006</td>
<td>15</td>
<td>Le-Hung Vu, Fabio Porto,</td>
<td>v 1.5 - Final revised version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manfred Hauswirth</td>
<td></td>
</tr>
</tbody>
</table>

## Reviewers

<table>
<thead>
<tr>
<th>Partner</th>
<th>E-mail</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>FZI and SAP</td>
<td><a href="mailto:Grimm@fzi.de">Grimm@fzi.de</a></td>
<td>+49 (721) 965-40</td>
</tr>
<tr>
<td>ISOCO</td>
<td><a href="mailto:slosada@isoco.com">slosada@isoco.com</a></td>
<td>+34 (913) 349-797</td>
</tr>
</tbody>
</table>
# PROJECT CONSORTIUM INFORMATION

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Galway | NUIG | Dr. Sigurd Harand  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
E-mail: sigurd.harand@deri.org  
Tel: +353 91 495112 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. BankInter,  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmtnez@bankinter.es  
Tel: 916234238 |
| British Telecommunications Plc. | BT | Dr. John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
Email: karl.aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council | Essex | Mary Rowlatt,  
Essex County Council,  
PO Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom  
Email: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik | FZI | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe  
Germany  
E-mail: abecker@fzi.de  
Tel: +49 721 96540 |
| 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 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Contact Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOG SA</td>
<td>Christian de Sainte Marie</td>
</tr>
<tr>
<td></td>
<td>9 Rue de Verdon, 94253, Gentilly, France</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a></td>
</tr>
<tr>
<td></td>
<td>Tel: +33 1 49082981</td>
</tr>
<tr>
<td>inubit AG</td>
<td>Torsten Schmale, inubit AG</td>
</tr>
<tr>
<td></td>
<td>Lützowstraße 105-106, D-10785 Berlin, Germany</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:ts@inubit.com">ts@inubit.com</a></td>
</tr>
<tr>
<td></td>
<td>Tel: +49 30726112 0</td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>Dr. V. Richard Benjamins, Director R&amp;D</td>
</tr>
<tr>
<td></td>
<td>Intelligent Software Components, S.A.</td>
</tr>
<tr>
<td></td>
<td>Pedro de Valdivia 10</td>
</tr>
<tr>
<td></td>
<td>28006 Madrid, Spain</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a></td>
</tr>
<tr>
<td></td>
<td>Tel. +34 913 349 797</td>
</tr>
<tr>
<td>MDR Partners</td>
<td>Rob Davies, MDR Partners</td>
</tr>
<tr>
<td></td>
<td>8 St. Andrew Street, Hertford, Herts., United Kingdom, SG14 1JA,</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:rob.davies@mdrpartners.com">rob.davies@mdrpartners.com</a></td>
</tr>
<tr>
<td></td>
<td>Tel. +44 (0)208 8763121</td>
</tr>
<tr>
<td>Hanival Internet Services GmbH</td>
<td>Alexander Wahler, Hanival Internet Services GmbH, Kirchengasse 13/1a A-1070 Wien</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:wahler@niwa.at">wahler@niwa.at</a></td>
</tr>
<tr>
<td></td>
<td>Tel. +43(0)1 3195843-11</td>
</tr>
<tr>
<td>The Open University</td>
<td>Dr. John Domingue, Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a></td>
</tr>
<tr>
<td></td>
<td>Tel.: +44 1908 655014</td>
</tr>
<tr>
<td>SAP AG</td>
<td>Dr. Elmar Dorner, SAP Research, CEC Karlsruhe SAP AG</td>
</tr>
<tr>
<td></td>
<td>Vincenz-Priessnitz-Str. 1</td>
</tr>
<tr>
<td></td>
<td>76131 Karlsruhe, Germany</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a></td>
</tr>
<tr>
<td></td>
<td>Tel: +49 721 6902 31</td>
</tr>
<tr>
<td>Sirma AI Ltd.</td>
<td>Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
</tr>
<tr>
<td></td>
<td>Tel.: +359 2 9768 303</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Unicorn Solutions Ltd, Malcha Technology Park 1, Jerusalem 96951, Israel</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Pieter De Leenheer, Starlab- VUB, Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium</td>
</tr>
</tbody>
</table>
LIST OF KEYWORDS/ABBREVIATIONS

- CoDIMS-G - Configurable Data Integration Middleware System for the Grid.
- DBMS - Database Management System
- DHT - Distributed Hash Table
- EAI - Enterprise Application Integration.
- NFP - Non-Functional properties.
- P2P - Peer-to-Peer
- QoS - Quality of Service
- NFPs - Non-Functional Properties
- SWS - Semantic Web service
- UDDI - Universal Description, Discovery and Integration protocol
- WSD - Web service Description
- WSMO - Web Service Model Ontology
- wsmo4j - WSMO API for Java
- WSML - Web Service Model Language
- WSMX - Web Service Execution Environment
- QML - Quality of service Modeling Language
- WSLA - Web Service Level Agreement
- WSOL - Web Service Offering Language
# Table of Contents

1 Problem Statement 1  
1.1 Semantic Web Services Discovery Process 1  
1.1.1 The Notion of Discovery 1  
1.1.2 Service Discovery Preliminaries 2  
1.1.3 Notion of QoS in our QoS-enabled Service Discovery 3  
1.1.4 The Necessity of a QoS-enabled Web Service Discovery Engine 4  
1.1.5 Different Phases of Service Usage 5  
1.2 SWS Descriptions vs. Discovery Strategies 6  
1.2.1 Relationship between SWS Descriptions and Discovery Strategies 6  
1.2.2 Levels of Semantic Web Service Descriptions 7  
1.2.3 Discovery Strategy Issues 7  
1.3 Existing Web Service Discovery Approaches 8  
1.3.1 Centralized Discovery 8  
1.3.2 Hierarchical (UDDI) Discovery 8  
1.3.3 Decentralized/Distributed (P2P) Discovery 9  
1.3.4 Support for QoS-enabled Discovery 10  
1.4 Scope of Work and our Contributions 11  
1.4.1 A Centralized View on Discovery 11  
1.4.2 QoS Monitoring and Reporting Issues 12  
1.4.3 Dealing with Heterogeneity in QoS Attributes 12  
1.4.4 Summary of Contributions 13  

2 Discovery Component Specification 14  
2.1 The Interaction between the Discovery Component and Other Actors in the DIP Framework 14  
2.2 Architecture of the QoS-enabled Discovery Component 16  
2.3 Dependency on External Components 17  
2.3.1 Mediation Service 17  
2.3.2 WSMO Web Service Repository 19  
2.4 Discovery Component API Specification 19  
2.4.1 List of Supported APIs 19  
2.4.2 Representation of QoS Features in WSMO Goals and in Service Descriptions 21  
2.4.3 An Algebra for Web Service Discovery 24  
2.5 Detail Description of the Discovery Component and Implementation Issues 27  
2.5.1 The QoS-enabled Service Discovery Process 27  
2.5.2 QoS Trust and Reputation Management Model 28  
2.5.3 WSMO Repository Implementation 29  
2.5.4 Detailed Designs of the QoS Discovery Component 30  
2.5.5 Task List 31
3 Example Scenarios

3.1 Searching for the best Stock Information Service

3.2 Selecting the best Products/Services

3.3 Searching for the suitable Internet Service Provider

3.4 Searching for the best Computational Service

4 Proposed Extensions of the WSMO Model

5 Conclusion
# List of Figures

1.1 Discovery based on semantic service descriptions .......................... 1  
1.2 Discovery based on semantic service descriptions ......................... 3  

2.1 The interaction among the discovery component and other actors in the system .................................................. 14  
2.2 Message diagram of the interaction among various actors in the system ..................................................... 15  
2.3 Internal architecture of the QoS-enabled Semantic Web service discovery component ...................................... 18  
2.4 The main APIs for discovery and their Java method declarations ................................................................. 20  
2.5 The report submission API and its Java method declaration ........................................... 21  
2.6 The reputation information API and its Java method declaration ............................................................. 21  
2.7 The proposed extension for the WSMO conceptual model ......................................................... 23  
2.8 Representation of QoS information in a WSMO Web service .............................................................. 32  
2.9 Representation of QoS information in a WSMO Goal ................................................................. 33  
2.10 Example of a discovery execution plan ................................. 34  
2.11 Use case diagram ................................................................. 35  
2.12 System state diagram ................................................................. 36  
2.13 Operator design class diagram ......................................................... 36  
2.14 Discovery sequence diagram ......................................................... 37  
2.15 Advanced discovery sequence diagram ......................................................... 37  

4.1 The proposed extension for the syntax of the WSML language ........... 44  
4.2 The proposed extension for the WSMO4J API ............................... 44
1 Problem Statement

1.1 Semantic Web Services Discovery Process

1.1.1 The 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. A crucial step is for an agent to decide which service endpoint offers the relevant functionality it needs to fulfill 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 modeling service semantics, and is expressed in the WSML language.

Figure 1.1 depicts a Semantic Web service discovery scenario involving such semantically annotated Web services.

![Diagram of Semantic Web service discovery process]

Figure 1.1: Discovery based on semantic service descriptions

Provider and requester parties issue semantic descriptions of real-world services as annotation of Web service interfaces. These descriptions are input to the discovery process that decides whether a service offer is relevant for the request. The result of discovery are the Semantic Web services as a means to access the corresponding real-world services via a WSDL interface. In this sense, the requester party retrieves a set of references to semantic capability descriptions considered relevant, together with references to their associated Web services. The computational elements on the level of interface and semantic annotation, i.e. Web services and their semantic descriptions, represent real world business-level services within the scope of the machine’s information space.
1.1.2 Service Discovery Preliminaries

The service discovery approach to be adopted in WP4 agrees on the conceptual distinctions regarding service, Web service and Web service description.

The concept of a service is also named after concrete service, in [32], and corresponds to an actual set of tasks and messages necessary to accomplish a certain functionality. This includes list of operations, service parameter values, etc.

Obviously, a service provider would not advertise each possible concrete service, rather it abstracts a set of concrete services into abstract services, which are implemented by Web services. Finally, in order to allow users (or agents) to find a required abstract service, a machine-processable description of an abstract service is specified and advertised. The abstract service description corresponds to a Web service description. Examples of each of these concepts can be found in [32].

In order to keep with a single terminology, this document will refer to the above mentioned definitions as service, Web service and Web service description.

Similarly to the abstract service specification, users’ requests are equally abstracted into a concept view of required services. Indeed, a more abstract representation of users’ desires is expected to offer greater chance of matching with advertised abstract services within an automatic service discovery scenario. Such procedure, however, may lead to false matches as not all services corresponding to the service abstraction specified by the user necessarily provide the functionality expected by the user. We will discuss in section 1.1.5 that the selection phase of service discovery permits users to refine the set of Web services descriptions automatically chosen during service discovery, eliminating therein the false positive ones. In this context, a goal is a semantically equivalent Web service description expression formalizing requesters’ desires.

In this context, Web service discovery is the process of identifying Web services whose description matches with a given goal.

Different strategies have been proposed for approaching the Web service discovery matching problem [25, 7, 38]. In [23] these approaches are classified into three levels according to the effort required to annotate Web services and the expected matching accuracy result. At a simpler level, Web service discovery is keyword-based and considered as a support for richer matching. The remaining levels adopt a set theoretic approach and explore ontologies as formal, machine-processable representation of domain knowledge to represent Web services descriptions.

In a set theoretic approach, a Web service is described through the set of objects it may deliver. Similarly, a goal is described as the set of objects a chosen Web service is required to provide. Finally, the matching process in a Web service discovery corresponds to identifying the Web services whose descriptions intersect with those of a given goal, as shown in Figure 1.2.

In Figure 1.2, elements in the $C_g$ and $C_{ws}$ sets are concepts specified in the goal and Web services descriptions, respectively. Each concept in $C_{ws}$ may describe different Web services, leading, in case of a match, to multiple candidate Web services. Conversely, a Web service may use a number of concepts in its description. When a match occurs, matching web service descriptions are ranked according to their relevance with respect to the user goal.

This approach of matching Web services descriptions to user goals does not, however, directly map to the state transition model adopted in WSMO to describe Web

---

\[1\] Note that such a definition includes functional and non-functional objects
services capabilities. An in-depth discussion on this matter, as well as a proposal for encoding services modeled by "the objects it delivers" approach, is presented in D4.8 [17].

1.1.3 Notion of QoS in our QoS-enabled Service Discovery

In our perspective, a Web service is an electronic description of concrete services, which may offer the functionality of a software component, for example, a data backup service, or a real-life (business) activity such as book shipping. When we talk about such a concrete book shipping service, the concept of QoS may lead to two different understandings: the quality of the Web-based entry point offering the book shipping functionality, i.e., its availability, robustness, execution-time, etc., and the quality of the shipping service itself, e.g., timely delivery vs. shipping cost, etc.

The notion of QoS in our work is generic, ranging from application-level QoS properties to network or software-related performance features that a user expects from the service execution. Examples of application-level QoS parameters are the quality of the food delivered by a online-order pizza service or the timely delivery of a book. Typical network or software-related QoS parameters are the availability, response-time, execution-time, etc., of associated Web services.

The software/network-related QoS parameters are well defined for many Web services across various domains, whereas the definition of application-level QoS parameters might require special knowledge provided by domain experts. However, for certain applications in reality, the difference between application-level QoS and software/network-related QoS is less obvious, e.g., for computational Web services.

Our QoS-enabled service discovery performs the searching for Web service descriptions based on the semantic matching between whatever QoS parameters (and non-functional properties) expressed by the user in his/her goal against those advertised in the service descriptions. The concept of QoS in our Web service discovery process thus includes both the quality of the real-life business offering described by its associated Web service and the performance attributes of the Web-based software component providing the access point for that real-life service.
1.1.4 The Necessity of a QoS-enabled Web Service Discovery Engine

Web service technologies enable us to develop e-Science and e-Business programs from various existing components via the World Wide Web. This trend is becoming clearer over time [18]. For example, the Global Market Research IDC reveals that 80% of the companies in the US will begin a Web service project by the year of 2008. People are anticipating that these web-based components will enable a program to link to another program across the Web and form a Web of applications interwoven in functionality, according to Steve Holbrook, IBM program director for emerging e-business [18].

Along with the ever increasing of the number of Web services, their inherent distributiveness and the dynamic and heterogeneity of them lead us to think of Web service quality as an important issue, both to service clients and to service providers. For instance, the proposal of using Web services as a business solution to Enterprise Application Integration (EAI) advocates the selection of the best services, in terms of both functionality and QoS, to develop applications with the highest possible level of quality. This raises the necessity of a Web service discovery engine that is capable of discovering the most relevant Web services given the functional and quality requirements of the searching users.

The following scenario illustrates the usage of Web service technologies as the first step to the automatic integration of Web services in the business process, which clearly shows the necessity of a Web service discovery solution that is QoS-enabled. Just imagine that a company would need many services with different functionalities for its project. Due to the very high cost of developing or buying the whole software-hardware solution package as well as the great efforts that must be taken for the deployment and maintenance of these services, this company can rely on certain existing Web services provided by many service providers, e.g., software companies, to acquire the needed functionality for its business and only pays the service providers for using these services during the time of the project. Some examples of such Web services are:

- Computational services capable of doing large-scale data mining and analysis. These services could be used for the analysis market statistics, predict future business opportunities and possible customer behaviors, etc.

- A temporarily secured backup data storage service with high-capacity and ultra-speed.

- A service providing up-to-date stock-market information, which produces data for related departments or for some internal analysis software systems of the company. This information will then be used for evaluating, discovering market trends and proposing appropriate future strategies.

- A Web service providing access to an already installed and configured, easy-to-use Internet-based video-conference systems.

- A trouble shooting/help-desk Web service for solving common problems in using various materials, Internet-related problems, as well as for offering advice and expert knowledge in various fields upon requests.

\[\text{Examples with more detailed information could be found in section 3}\]
In the above examples, given advertised prices of these services and the limited budget of the companies, it is particularly important for the service clients to discover the best (most-suitable) Web services to use, e.g., the services should be secured, available almost everywhere and at anytime, have acceptable response time, robust against the most popular errors as well as capable of handling possible exceptions during their execution, etc. In such a situation the existence of a QoS-enabled service discovery engine would be highly important and irreplaceable.

1.1.5 Different Phases of Service Usage

There are different phases in the whole cycle of usage of Web services around discovery. Based on [23], we distinguish between goal definition, Web service (description) discovery, service selection (definition) and service delivery.

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., specialization 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. **Web service (Description) Discovery** is based on comparing the semantic description of a requested abstract service against those of provided abstract services in terms of relevance, as described in Section 1.1.1 and Section 1.1.2.

3. **Web service Selection (Definition)** 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 process 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 goals and Web service capabilities. In this phase, the generalization 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. **Web service Delivery** comprises any steps required for the actual invocation and consumption of the service selected (defined) 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.

In the work on discovery within WP4 we plan to sketch an overall architecture that covers all of the phases described above. However, we see the focus to be set on the service discovery phase when it comes to detailed conceptualization and implementation. 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 selection (/definition) 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 WSMO or semantic service descriptions in general. We argue that the main actor in the service selection (/definition) phase is the agent, acting as requestor or provider, which invokes the interfaces of potential Web service candidates identified through discovery.

1.2 SWS Descriptions vs. Discovery Strategies

1.2.1 Relationship between SWS Descriptions and Discovery Strategies

Discovery of SWS depends on two issues: (1) the level and type of Web service descriptions and (2) the possible discovery strategies. The level and type of Web service description basically determines the maximum possible information which can be found by any discovery strategy, i.e., you can only discover what has been described, together with results based on inference, etc. which again is determined by the quality and level of detail of Web service descriptions.

Though the discovery strategies of course depend on the available information, they are independent of that to a large degree as the quality of the description basically determines an “upper bound” of the discovery process. But the discovery strategy can be defined at any level below this upper bound. For example, given very detailed Web service description, the discovery strategy could only index one attribute of the description and offer full-text search on the attribute values or it could index certain sub-structures and then support conjunctive queries on these attributes.

So the discovery strategy and the Web service descriptions can actually operate on different levels of abstractions which are largely defined by application requirements. Additionally, the discovery strategy critically depends on the level of distribution of the underlying information and what cooperation models are used. For example, a centralized registry can easily support complex reasoning models, whereas the support for that decreases rapidly with the level of distribution of information. Current reasoning approaches are centralized and require exhaustive global knowledge. Distributed reasoning, especially in highly distributed environments, suffers from the lack of such knowledge which, however, is essential for the interpretation and reasoning tasks. At the moment, no usable off-the-shelf approaches exist which we could have used as distributed reasoning is an area of ongoing research. Due to these problems, we have to trade distribution (scalability) against the level of functionality and vice versa.
The abstraction level in the descriptions is largely orthogonal to the discovery (matching) process. The process does not necessarily need to use all of the available details given in a WS description. Thus it may possibly become more scalable but less accurate, for instance given a lightweight semantic description by means of a (complex) concept. To find the right discovery strategy for the given type of data can be seen analogous to the problem of query optimization in database system, where the question is to choose different types of joins for specific tables to obtain an optimal result by some metric.

1.2.2 Levels of Semantic Web Service Descriptions

According to WSMO discovery model [23], a Semantic Web service could be described at different levels: syntactic, light-weight semantic, and heavy-weight.

1. At the syntactic level, service descriptions are provided in the terms of advertisements along with the keywords for categorization.

2. At the light-weight or simple semantic level, Web services are represented as sets of objects. A Web service can be seen as computational object which can be invoked by a client. The resulting execution of the Web service generates (w.r.t. a set of input values provided by the client) certain information as an output and achieves certain changes of the state of the world. An output as well as an effect can be considered as objects which can be embedded in some domain ontology, that means a formal conceptualization of the problem domain under consideration.

3. At the heavyweight or rich semantic level, we could describe how outputs and effects created by a service execution actually depend on the concrete input provided by the user when invoking a service. That means we would consider a service as a relation on an abstract state-space and capture the functionality provided by a service in these terms. Here, we would consider services input, output, precondition, assumption, postcondition and effect of a service.

1.2.3 Discovery Strategy Issues

A fundamental assumption is that there is no “one size fits all” discovery strategy. The best strategy will be determined by application requirements. Also multiple discoveries can co-exist in one system. It is then up to the system to decide which one would be most suited for certain types of queries.

The discovery strategy most prominently depends on:

1. The degree of distribution (e.g., Web service descriptions, ontologies, etc.) of the underlying system holding the descriptions;

2. the level of scalability (in terms of, e.g., number of service descriptions) that should be achieved;

3. the degree of semantic heterogeneity, i.e., how many ontologies exist and how often is mediation between them required; and
4. the discovery evaluation strategy, i.e., how to evaluate the efficiency of a discovery strategy in terms of performance, completeness, etc.

1.3 Existing Web Service Discovery Approaches

1.3.1 Centralized Discovery

As mentioned in Section 1.1, Web service (description) discovery is based on ontological descriptions of services in WSMO using the WSML language as the underlying knowledge representation formalism. The way in that ontological descriptions are usually processed is that of using automated reasoning techniques that enable the derivation of implicit knowledge from explicit knowledge. Such techniques support for a sophisticated realization of the discovery process in deciding whether a service offer is relevant for a service request.

The basic idea of the logics-based matchmaking is based on the notions of concrete service and abstract service, as described in [32]. A concrete service represents a particular business transaction fixing all the service parameters in detail. An example of a concrete service taken from the traveling domain is “selling a flight ticket from Frankfurt to Galway at a particular date and time for 50 Euro”, offered by some travel agency [17]. However, the travel agency also provides other concrete services, which vary depending on the cities, date or price. Instead of listing all these possibilities in a service description, such a travel agency would rather describe an abstract service, which approximates the set of all concrete services. An example for an abstract service would be “selling flight tickets between cities in Europe”.

In a centralized discovery approach, abstract services descriptions are published in a central semantic Web service repository. Access to such descriptions is provided by a Web service discovery component.

In order to find desired services, users send a Web service discovery request to the discovery component that proceeds to a semantic matching of the user goal against published abstract services descriptions.

In the coordination of our work and the functionality-based service discovery approach described in D4.8 [17], semantic service descriptions describe abstract services, which maps to a modeling of service capabilities as service concepts that can have many instances reflecting the different configurations of service parameters. Two such conceptual descriptions match if they have concrete services in common, i.e., if they allow common parameter configurations. In terms of logical matchmaking, this is verified by checking the satisfiability of the conjunction of the goal and capability concepts.

1.3.2 Hierarchical (UDDI) Discovery

For Web services to be meaningful there is a need to provide information about them beyond the technical specifications of the service itself. Central to UDDIs purpose is the representation of data and meta-data about Web services. A UDDI registry, either for use in the public domain or behind the firewall, offers a standard mechanism to classify, catalog and manage Web services, so that they can be discovered and consumed. Whether for the purpose of electronic commerce or alternate purposes, businesses and providers can use UDDI to represent information about Web services.
in a standard way such that queries can then be issued to a UDDI Registry at design-
time or run-time that address the following scenarios:

- Find Web services implementations that are based on a common abstract inter-
  face definition.
- Find Web services providers that are classified according to a known classification
  scheme or identifier system.
- Determine the security and transport protocols supported by a given Web service.
- Issue a search for services based on a general keyword.
- Cache the technical information about a Web service and then update that in-
  formation at run-time.

These scenarios and many more are enabled by the combination of the UDDI in-
formation model and the UDDI API set. Because the information model is extremely
normalized, it can accommodate many different types of models, scenarios and tech-
nologies. The specification has been written to be flexible so that it can absorb a
diverse set of services and not be tied to any one particular technology. While a UDDI
node exposes its information as an XML Web service, it does not restrict the tech-
nologies of the services about which it stores information or the ways in which that
information is decorated with meta-data.

UDDI is a potential candidate for non-semantic (keyword-based) discovery solu-
tion. In general, by semantically enhancing UDDI, it must be possible to provide
semantic discovery services on top of UDDI, e.g. [6]. However, the scalability of the
distribution concept used in UDDI has to be investigated. The discovery tasks in
DIP do not cover the publishing of Web services. Therefore, the technical feasibility
of a semantically-enhanced UDDI discovery solution depends on the availability of a
semantically-enhanced UDDI publishing solution.

1.3.3 Decentralized/Distributed (P2P) Discovery

The emerging number of services in various private and public UDDI registries advo-
cates for the demand of highly scalable and efficient service discovery architectures.
As with other information retrieval systems, this goal of highly scalability and effi-
ciency could only be achieved with decentralized service discovery models relying on
a global P2P system as its service repository network. A decentralized semantic Web
service discovery architecture requires the cooperation among various registries when
searching for a specific Web service that fulfills all functional and quality requirements
of users. Generally, service advertisements are published in a decentralized (or P2P)
network of registries by various providers, and users can query for services with cer-
tain functionalities and required QoS levels using any registry as their access point.
The P2P-based registries then take care of routing the request to the peer(s) that can
answer it. The results will be returned to the user and this user may invoke one of the
found services. In reality, the companies who manage the search engines would collect
service description information in various public/private registries and provide users
with semantic-based searching capabilities.
The first important issue in designing such a system is which type of overlay network, i.e., structured or unstructured, will be used as its decentralized service description repository. The second issue is how to distribute and index these semantic descriptions of various services among search engines so as to minimize the search cost during the discovery phase.

Existing work targeting this goal of P2P-based Web service discovery includes the following relevant proposals. METEOR-S [39] and HyperCup [34] base the distribution of semantic Web service descriptions on a classification system expressed in service or registry ontologies. In these approaches, the choosing of a specific registry to store and search for a service advertisement depends on the type of the service, e.g., business registry is used for storing information of business-related services. In our opinion, these solutions are good in terms of organizing registries to benefit service management rather than for the service discovery itself. Though it is relatively simple when publishing and updating service description information based on their categories, it would be difficult for users to search for certain services without knowing details of this classification, let alone come up with such a common service or registry ontology.

WSPDS [22] uses an unstructured P2P underlying network as the P2P-based service repository, which would be not very highly scalable and efficient in terms of searching and updating costs. [35] indexes service description files (WSDL files) by a set of keywords and uses a Hilbert-Space Filling Curve to map the n-dimensional service representation space to an one-dimensional indexing space and hash it onto the underlying DHT-based storage system. However, the issue of characterizing a semantic Web service description in a multi-keyword form in order to support semantic discovery of services has not yet been mentioned in this work. Emekci et al [16] suggest to search services based on their execution paths expressed as finite path automata which we consider less important. The main reason is it is difficult to use as primary selection condition in queries since users are required to know the execution of their required services.

1.3.4 Support for QoS-enabled Discovery

In the e-Business (and e-Science) application domains, besides the key issue of discovering the most relevant services meeting the functional requirements of users, the service clients also would like to discover services which best meet their requirements in terms of QoS, i.e., performance, throughput, reliability, availability, trust, etc. Thus QoS-based Web service selection and ranking mechanisms will play an essential role in service-oriented architectures, especially when the semantic matchmaking process returns lots of services with comparable functionalities. We define a QoS-enabled service discovery as the process of discovering Web services according to user requirements, which would include both functionality and quality of service. In this type of discovery, the QoS requirement part in user queries is optional and used as an additional criteria for ranking the results.

Although the traditional UDDI standard [24] does not refer to QoS for Web services, many proposals have been devised to extend the original model and describe Web services’ quality capabilities, e.g., QML, WSLA and WSOL [27, 14]. The UX architecture [9] suggests using dedicated UX servers to collect feedbacks of consumers and then predict future performance of published services. [8] proposes an extended implementation of the UDDI standard to store QoS data submitted by either service
providers or consumers and develop a special query language (SWSQL) to manipulate, publish, rate and select those data from repository. According to Kalepu et al. [21], the reputation of a service should be computed as a function of three factors: different ratings made by users, service quality compliance and its verity (i.e., the changes of service quality conformance over time). However, these solutions have not yet mentioned the trustworthiness of QoS reports produced by various users, which is important to assure the accuracy of the QoS-based selection and ranking results. [26] rates services computationally in terms of their quality performance with QoS information provided by monitoring services and users. The authors also employ a simple approach of reputation management by identifying every requester to avoid report flooding. In [16], services are allowed to vote for quality and trustworthiness of each other and the service discovery engine utilizes the concept of distinct sum count in sketch theory to compute QoS reputation for every service. However, these reputation management techniques are still simple and not robust against various cheating behaviors, e.g., collusion among liars with varying actions over time. Consequently, the quality of ranking results of those discovery systems will not be assured if there are lots of colluded dishonest users trying to boost the quality of their own services and badmouth about the others’ ones. [13] suggests augmenting service clients with QoS monitoring, analysis and selection capabilities, which is a bit unrealistic as each service consumer would have to take the heavy processing role of both a registry and a reputation system. Other solutions [37, 33, 29, 30, 28] use mainly third-party service brokers or specialized monitoring agents to collect performance of all available services in registries, which would be expensive in reality. Though [28] also raises the issue of accountability of Web service Agent Proxies in their systems, the evaluation of trust and reputation for these agents is still an open problem.

1.4 Scope of Work and our Contributions

1.4.1 A Centralized View on Discovery

In the implementation of the discovery component, we take a centralized rather than a fully distributed approach for the following reasons: First, current work on discovery and matchmaking is based on a rather simplistic view and leaves out a lot of the complexity of real-world business discovery tasks. In order to make Semantic Web services a reality, we regard it as crucial to provide a functionality that addresses the full complexity rather than focusing on scalability issues.

Second, actual Web service discovery will, for the foreseeable future, take place in Business Services Networks, i.e., not at the full scale of the World Wide Web but rather clusters with a guaranteed quality of service and security functionality.

Third, the amount of annotation data about Web services in such Business Services Networks will rather be in the size of hundreds of Megabytes, which can be easily represented in a centralized repository.

Fourth, mechanisms of load-balancing and redundancy (e.g., redirecting discovery requests to one out of many identical discovery engines) seem to be sufficient.

In this context, although dealing with centralized repositories of Web service descriptions, we transparently (with respect to the user) cope with a potential increase
in the number of publications by parallelizing the evaluation of the discovery matching algorithm.

1.4.2 QoS Monitoring and Reporting Issues

In order to evaluate the actual QoS levels of published Web services, we suppose that the QoS-enabled discovery component have statistics of the QoS parameters of some specific Web services, which are obtained from a few trusted third-parties on a periodical basis. In reality, this task of QoS monitoring could be done with the support of specialized monitoring software, which could be provided by companies like AlertSite [1], Dot-Com Monitor [2], and Empirix [3], etc.

Additionally, we expect that the users who have requirements for more sophisticated quality parameters have their own way of monitoring and measuring the perceived quality. An alternate solution is that providers release their own QoS measurement software, along with the advertisement of their corresponding services, as a component that could be installed in the client’s machine to evaluate the performance of the executed services.

We also assume that service users are willing to report their experience, i.e., the QoS parameter values they actually perceive while using a service, to our QoS-enabled discovery component. This is realistic because of the following reasons. First, by reporting these QoS values, the users help the discovery component to evaluating performance of services more accurately, which will allow them to have more precise results for their future service queries. Therefore, they can choose the most appropriate services to use w.r.t. various quality requirements and budget limitations. Second, in online service markets where users have to pay for the consumed services, those service users giving feedback on their experienced quality of service may be beneficial from certain side-payment mechanisms, e.g., have reduced subscription prices, from the providers or from the discovery component.

1.4.3 Dealing with Heterogeneity in QoS Attributes

QoS-enabled Web service discovery encompasses the problem of dealing with heterogeneity in matching QoS attributes in goals and those used in Web service descriptions. One may envisage different scenarios in which QoS attributes are defined with respect to used vocabulary. In one scenario, one may consider a standard set of QoS properties defined by an agreed committee, or according to an agreed upon ontology. In this scenario, matching QoS properties in goals and Web service descriptions (WSD) resumes to concept comparison.

In a second scenario, WSD and goals express QoS parameters according to heterogeneous QoS ontologies. In this case, one could first homogenize terminologies, possibly using mediators, and in the sequel compare concepts, as in the first scenario.

We assume the first scenario for the first prototype of our QoS-enabled discovery component whereas leaving the issue of heterogeneity on QoS concepts to be dealt with during the second prototype of our work. For the second version we will use the mediation service for translating and matching QoS concepts from different ontologies.
1.4.4 Summary of Contributions

In summary, the QoS-enabled discovery component will be developed as an independent service offering the semantic-based discovery facilities for semantic Web services to various DIP-WSMX components and service clients via an WSMX-compatible API. The main contributions expected from our discovery component are manifold:

- Semantic service matchmaking: since we allow rich semantic description of QoS in our approach, our QoS-enabled service discovery supports semantic matching of user requests to Web service descriptions. This matching process is modeled through an expression within a specified algebra, of which the match function is an operation, permitting its implementation using different semantic matching algorithms.

- QoS modeling: QoS attributes are integrated into WSMO framework using Non-Functional properties constrained by a QoS ontology. This model also allows expressing predicates over QoS attributes and specifying various conditions for achieving a certain QoS level.

- QoS values capturing and updating: manages quality of service data that are collected during and after Web service invocations.

- Ranking of Web services based on QoS compliance: when matching QoS requirements in goals with those in Web service descriptions, we compute an associated ranking value for each Web service description according to its relevance to the user’s QoS requirements, taking into account the evaluated reputation information of the Web service provider as well. The final search result is a list of Web service descriptions sorted in descending order of their associated ranking values.

- Managing of trust and reputation issues adequately: the reputation-based management mechanism will identify cheating attacks coming from competitive service providers and use it to determine provider’s trustworthiness.

- Designing and implementing a new Web service Description Repository: our work includes a Web service Description repository framework capable of storing WSML files into commercial DBMS and pushing-down QoS predicate evaluation into the DBMS, minimizing query evaluation cost;

- Introducing a new and efficient QoS-enable Semantic Web service discovery framework: the service discovery model is designed and implemented based on a query processing system architecture in which operations are modeled by algebraic counterparts allowing for query optimization and parallelization of operators execution.

---

3This compatibility ensures that we can easily integrate our discovery component into the WSMX architecture for possible deployment, testing and usage in some example scenarios.
2 DISCOVERY COMPONENT SPECIFICATION

2.1 The Interaction between the Discovery Component and Other Actors in the DIP Framework

A high level integration architecture view of the discovery component in the DIP framework is depicted in Figure 2.1.

Figure 2.1: The interaction among the discovery component and other actors in the system.

As illustrated in Figure 2.1, the operation of the Semantic Web service Discovery process in the whole service usage life-cycle comprises the following interactions:

- A service user can query for services with certain functionalities and required QoS levels by submitting a goal (initial goal defined in the goal definition phase) to the Semantic Web service Discovery Service (discovery component interface). Normally, users can achieve this via the help of DIP-WSMX tools (1a, 1b and 2a), or use the discovery component APIs directly (2b) (expert users such as computer agents, developers). Users expect the result as a set of Web service descriptions matching their specified goals.

- The QoS-enabled Web service discovery process will perform the necessary semantic Web service discovery algorithm to find out and return a ranked list of the best matched Web service descriptions to the user. During this step, the discovery component would need to use Web service description information in the centralized WSMO service registry (2c) or the mediation service (2d) via their interfaces. The result set of Web services has to be further processed by the agent of the service requestor in the service definition phase.

- Users may invoke one of the found services (service definition phase). Again, other DIP-WSMX tools could be used to check for other compatibility issues, to perform the mediation steps, etc., and help users to select the most appropriate
services (1a, 1b, 3a) as well as to execute the services if necessary. Users can also invoke the Web services themselves (3b).

- Additionally, users can express feedbacks on the QoS they obtain from a service to the discovery component. The information from client feedback will be used to benefit future searches in the system. As this measurement and reporting of QoS require the users to have certain knowledge and expertise, we presume that the feedback collection could be done automatically by the DIP-WSMX service invoker components on behalf of novice users (4a). With expert users, QoS measurements and reporting can be done manually by the computer agents or the developers themselves (4b) via another report submission API.

- The discovery component deploys certain third-party trusted agents to monitor and collect QoS statistics of some specific Web services to benefit its QoS-enabled service discovery (5a). Specifically, the data collected from these trusted users would be used to evaluate the trustworthiness of other service providers and clients in the system. Service providers can also monitor QoS of their published services themselves and can query their QoS information from the discovery component to have an update view of their own service quality (5b).

The above interaction among various agents is also represented in the message diagram in Figure 2.2.
2.2 Architecture of the QoS-enabled Discovery Component

The internal architecture of the Semantic Web service discovery component is illustrated in Figure 2.3 and consists of the following sub-components:

1. **Query Analyzer**: This module receives a user’s goal (WSMO Goal), extracts relevant information, i.e., required functional and QoS specifications and produces an internal representation. It then passes the control over to the query optimizer.

2. **Query Optimizer and Execution Manager**: Given an internal representation of the user’s goal, this sub-component performs two tasks. First, it calls the functionality-based service discovery component, as described in Deliverable D4.8 [17], to get a list of WSMO service descriptions satisfying the required functional features of user’s goal\(^1\). This list of service descriptions is then stored in a temporary data store (Internal WMSO Service Repository) of the QoS-enabled discovery component. Second, the Query Optimizer and Execution Manager produces an equivalent operator tree, using the algebraic operators as defined in section 2.4.3. The operator tree specifies the operations to be evaluated and the dataflow to be followed. It is sent to various Semantic Query Engines that construct the corresponding operators in memory and starts their evaluation [31]. In the context of a centralized Web service discovery approach, the query optimizer can speedup the query execution by parallelizing the *match* (Section 2.4.3), operator. In this case, a fragment of the operator tree containing parallelized operations is scheduled to run in multiple nodes. The CoDIMS-G system [31] supports this query execution model and provides necessary communication control and dynamic load-balancing. The data sources for the replicated Semantic Query Engines to work on are the QoS Data Repository and the Internal WMSO Service Repository, as explained later. Since the semantic matchmaking is an expensive task and the discovery component could be used by different users at the same time who may have various QoS requirements for a variety of applications, this optimization is particularly important for the design of our discovery component.

3. **Semantic Query Engines**: Once a query execution plan has been produced, it is time to execute the query. Execution starts by instantiating the query execution plan as an in memory operator tree. Each scheduled node will run a Semantic Query Engine that receives an allocated fragment of the query plan. Specifically, each of these engines is in charge of performing the selection and ranking of services based on the restriction on QoS properties. These restriction operators eliminate undesired Web service descriptions by evaluating predicates over QoS values. Valid Web service descriptions from the Internal WMSO Service Repository, with respect to the restriction predicate, feed the *match* operator for a semantic predicate evaluation. Different implementations of the *match* operator may be envisaged according to the discovery strategies adopted (see Section 1.2). During execution, the mediation service may also be accessed for dealing with the translation among different QoS ontologies.

\(^1\)This interaction is done via the same *discover()* API which is implemented in the functionality-based discovery component, as specified in the Deliverable D4.8 [17].
4. Internal WSMO Service Repository: This is a repository storing the list of WSMO Web service Descriptions fulfilling functional requirements of the users and to be selected and ranked according to their QoS capabilities. This repository acts as an additional data source for the Semantic Service Query Engine during the QoS-enabled service discovery process.

5. QoS Reputation Mechanism: This module maintains and updates the reputation information of different services/providers on their QoS conformance over time, given the data collected from various QoS reporters (those users who produce feedback: computer agents, developers, DIP-WSMX components, trusted agents, etc.). The reputation mechanism contacts the Internal WSMO Service Repository and the WSMO Service Registry Interface to retrieve and update the service descriptions to use in its periodical evaluation of QoS of various Web services. QoS reputation information of services estimated by this subcomponent is then written to the QoS Data Repository.

6. QoS Data Repository: We use this repository specially for storing the feedback of QoS reporters, the reputation information of all service providers as well as the predicted QoS information of all QoS-aware Web services, i.e., those services with QoS information in their descriptions, in the system. The QoS Data Repository acts as an additional data sources for the Semantic Service Query Engine during the QoS-enabled service discovery process.

Some main UML design diagrams of the discovery component are included in Appendix 2.5.4 of the deliverable.

2.3 Dependency on External Components

We identified dependencies of the discovery component on two external components: one for data mediation between ontological vocabularies and one for storage of semantic service descriptions.

2.3.1 Mediation Service

One could assume that Web service capabilities and goals are described using the same terminology. Then no data mediation problem exists during the discovery process. However, it is unlikely that a potentially huge number of distributed and autonomous parties will agree before-hand 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 data mediation service in place. Given the previous assumption, we can optimistically assume that a mapping has already been established between the used terminologies, not to facilitate our specific discovery problem but rather to support the general information exchange process between these terminologies.

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

Therefore, with respect to mediation, service discovery requires only data mediation on ontologies and their instances. Service definition requires also protocol and process mediation.

In WSMO version 1.0 a goal description contains only post-conditions and effects whereas it contains neither assumptions and pre-conditions nor the interface (choreography and orchestration) of the requester [10]. This means in WSMO version 1.0 the
goal and Web service descriptions are not symmetric and the goal description is less precise (contains less information) than Web service capability description. Especially, it is not possible to check during service discovery if protocol and/or process mediation is possible, because the interface required by the requestor is not contained in the goal description.

In WSMO version 1.1 [11] the situation is different. The goal description contains the requested service capabilities as well as the requested service interfaces. This means the goal and Web service descriptions are symmetric and the goal description can be as precise as Web service description. It is therefore theoretically possible (at least for an exact match) to check for protocol/process mediation. The question whether it conceptually makes sense to incorporate this step in the service discovery phase is a subject for further discussion.

2.3.2 WSMO Web Service Repository

The Web service discovery component is supported by a WSMO repository. The latter manages the storage and retrieval of Web service’s descriptions, WSMO ontologies and mediator’s descriptions. Access to a repository implementation is defined by the wsmo4j interface [5]. Retrieving entities from a repository is obtained by calling a retrieve method from WSMORegistry class, having a single string parameter. This basically provides developers with great flexibility in choosing the language to be used in their repository implementation, although limiting interoperability with client applications. In particular, the implementation of a WSMO repository as a layer on top of a DMBS which provides rich query expressivity enables us, for instance, to push-down restriction operators to the database server. This is extremely relevant for filtering out Web service descriptions which QoS values do not qualify users’ requirements.

During the evaluation of a semantic match operator, it may be required to retrieve ontologies. The WSMO repository, in addition to Web service descriptions, manages the storage of ontologies. The match operator, in this scenario, would access the repository to retrieve fragments of used ontologies necessary to support the semantic matching between the goal and Web service description. Similarly, if mediation is required, the repository is accessed for obtaining the URL of used mediators.

2.4 Discovery Component API Specification

To facilitate the usage of our QoS-enabled discovery component as an independent service or as a component to be integrated to the WSMX framework [4], we offer a list of APIs, which fully conforms to the WSMX’s info model. In the following sections, we will use pseudo-code for the convenient explanation of the meanings of various parameters in each API and provide a reference Java declaration for the corresponding method using existing wsmo4j packages and classes.

2.4.1 List of Supported APIs

As the basic API to the discovery functionality we intended a WSMO Goal as the main input and a list of WSMO Web service elements as the main output parameters. The basic APIs for the discovery are detailed as in Figure 2.4.
The main APIs for discovery

```
discover(IN: goal,
    OPTIONAL IN: rankingOntology,
    OUT: OrderedListOf{Pair{webServiceDescription, serviceInterface},rankingInstanceID});
```

```
discover(IN: goal,
    OUT: OrderedListOf{webServiceDescription});
```

```
discover(org.wsmo.goal.Goal goal, org.omwg.ontology.Ontology rankingOntology);
```

```
List<org.wsmo.service.WebService> discover(org.wsmo.goal.Goal goal);
```

Figure 2.4: The main APIs for discovery and their Java method declarations.

The `goal` parameter contains requested capabilities as well as the required QoS, which are all expressed by WSMO ontology concepts as represented in Section 2.4.2.

As the search results may contain lots of Web services with different matching levels to the user query, another step of ordering the search results based on their relevance to the requestor’s goal is necessary. The optional `rankingOntology` parameter is used to specify this information. With the use of an ontology to describe various user’s preferences, ranking criteria, matching functions, etc., the discovery engine enables highly flexible and extensible ways of filtering and ranking the immediate results to obtain the most relevant services for users’ needs. The development of sample ranking ontologies would be done in the implementation stage of our discovery component. Basically, we envisage the following information to be used in this parameter:

1. The relevance of each term, e.g., each QoS parameter, in the user’s query (goal).
2. The required level of matching between each term of the query and the corresponding concept in the Web service description, i.e., the semantic similarity threshold between an advertised service’s functional or quality property and that of the requested service.
3. The matching expression to be used in the ranking process.

The output parameter of the discovery process is a list of Web service descriptions matching user’s `goal` and ranked according to the provided criteria in the `rankingOntology` parameter. As each WSMO Web service description has many interfaces with different non-functional (QoS) properties, only some of which are relevant for the user query, the result list also contains the identifier to the ontological instance describing the ranking value of each pair of the corresponding matching Interface (`serviceInterface`) and its associated Web service Description (`webServiceDescription` variable).

The QoS-enabled service discovery component also offers additional APIs for service clients as well as for other modules in the system. The API for clients to submit their experience on QoS of certain services is in Figure 2.5.

In the above declarations, `serviceID` and `serviceInterfaceID` are respectively the identifiers of the Web service and the corresponding service interface on which the client wants to submit her report. The feedback on QoS of a Web service is a list of
FP6 – 507483
Deliverable 4.17

The report submission API

```java
submitQoSFeedback(IN: serviceID, IN: serviceInterfaceID,
                   IN: timeStamp, IN: ListOf{qosReportedValues});

void submitQoSFeedback(org.wsmo.common.Identifier serviceID,
                        org.wsmo.common.Identifier serviceInterfaceID,
                        Time timeStamp, List<org.omwg.ontology.Instance>);
```

Figure 2.5: The report submission API and its Java method declaration.

$qosReportedValues$ objects each of which is an instance of a concept in the QoS domain ontology reflecting the perceived value of the corresponding QoS parameter that the user observed at a specific time point $timeStamp$. The feedback could be generated and collected automatically by the system or submitted manually by users.

For cooperating between the discovery component and other components in the systems, we also give another API to provide the reputation information of a service provider during a time period, as in Figure 2.6.

The reputation information API

```java
getProviderReputation(IN: providerID,
                       IN: startTimePoint, IN: endTimePoint,
                       IN: ListOf{qosParameter},
                       OUT: ProviderReputation);

float getProviderReputation(org.wsmo.common.Identifier providerID,
                            Time startTimePoint, Time endTimePoint,
                            List<org.omwg.ontology.Concept>);
```

Figure 2.6: The reputation information API and its Java method declaration.

Given the identification of a provider ($providerID$), this API returns the general evaluated reputation of this provider ($providerReputation$) in providing the quality attributes specified in the list $ListOf$qosParameter during a time period (from $startTimePoint$ to $endTimePoint$). Note that one could use this API to get reputation of a provider in providing each individual quality parameter by putting only one concept in the parameter $ListOf$qosParameter and from this compute the aggregated reputation of the provider on a list of different quality attributes using her own method.

### 2.4.2 Representation of QoS Features in WSMO Goals and in Service Descriptions

Regarding QoS properties of Web services, we need to provide a semantic description for the following important information:

- The basic knowledge about the QoS parameters, e.g., their parameter names, textual descriptions, measurement units, requirements on their parameter values, etc.
• The conditions to achieve a certain QoS level, e.g., the price of the service, the minimum network connection and system requirements, the maximal number of requests the service can handle, the location of users, etc.

The service providers will publish their service advertisements, including the information on QoS properties, into a service registry. From a user’s side, similar information can also be specified in her queries while searching for the needed Web Services.

Regarding the WSMO specification, we have decided to encode QoS properties of the Web services as Non-Functional Properties (NFPs) of a WSMO Interface that is connected to a WSMO Web service or a WSMO Goal. This is the most appropriate place due to the following two reasons:

• our QoS extension is of the same nature as the notion of NFPs in the WSMO model, i.e., we also mention of the performance, security, accuracy, etc., of Web services. Our QoS properties could also be used in parallel with existing QoS attributes proposed by WSMO. Therefore, it is consistent to consider QoS parameters as NFPs.

• a Web service possibly has many interfaces, each of which corresponds to a Service Level Agreement (SLA) guaranteed to be delivered under specific conditions, e.g., price. These SLAs are advertised by the service provider during the service publishing phase and contain various QoS properties of this Web service. Symmetrically, a user can specify her quality requirements in the requestedInterface in the WSMO Goal she submits to the discovery component. This is consistent with other QoS specification and description model in reality.

Currently, the latest WSMO conceptual model only enables the declaration of simple values for non-functional properties, including quality-related parameters such as availability, accuracy, etc [12]. Therefore, it is not expressive enough for the specification of complex QoS parameters and for the description of various conditions of each Service Level Agreement of a Web service. In reality, information on the QoS levels claimed by the Web service providers as well as the QoS requirements specified by various service users can be sophisticated and depend on many complex conditions. Based on real-world requirements and the existing QoS specification standards, e.g., QML, WSLA and WSOL [27, 14], we envisage that the description of user’s QoS requirements and providers’ QoS advertisements using the WSMO model should be expressive enough to ensure the feasibility of the development of a full-fledged QoS-enabled service discovery component for realistic applications. Figure 2.7 shows our proposed extensions for the WSMO conceptual model in order to achieve that purpose. The syntax of the WSML language as well as associated tools should also be updated towards these changes as well. Please refer to the Section 4 for the details information about this proposal.

Please note that simple predefined NFPs of the WSMO model such as wsml#security, wsml#networkRelatedQoS, etc., can still be used in Web service and Goal Descriptions in the case service providers/users have no special needs for the describing of complex quality attributes.

Given the extended WSMO model, the non-functional properties of each Interface of a WSMO service description or WSMO goal consist of a set of constraints on the QoS
The proposed extension for the WSMO conceptual model

```
Class nonFunctionalProperties
  ...other existing properties...
  hasQoSParameters type QoSParameter

Class QoSParameter
  hasQoSConstraint type axiom
  hasQoSCondition type axiom
```

Figure 2.7: The proposed extension for the WSMO conceptual model

instances of the concepts imported from QoS domain ontologies. Each QoS constraint is paired with the axiom defining the conditions to be fulfilled in order to achieve that specific QoS level.

The usefulness of this representation model in practical applications is illustrated in the following simple yet realistic example. A financial-information service provider offers a Web service named `euStockMarketMainIndexes()` to retrieve the most important indexes information of the European stock market. We suppose that this service is accessible via the two following interfaces:

- The first interface is of free and therefore has a quite low availability level with a lower-bound 0.99, i.e., the probability that the service to be available given all conditions fulfilled is at least 0.99. Furthermore, the service is only accessible in the TechGate building in Vienna city, and the user can not use this service for the first day of the month.

- The second service interface requires the user to subscribe and pay for the obtained information. Consequently, its level of availability is much higher, at least 0.999999. Moreover, the produced information is assured to be as recent as the actual values in the stock market within 10 minutes, the service could be accessible anywhere anytime without any restrictions. However, to use the service via this interface, a valid credit card number of the user is required to ensure her payment of using this service.

Figure 2.8 illustrates the way we encode QoS parameters and associate QoS conditions for this service.

Supposed that a user needs to search for a service which provides main indexes of the EU stock market. She is currently in the Vienna International Airport with free wireless Internet access and would like to obtain the above information which should be as latest as 15 minutes compared to the real values. With the help of additional GUI tools, the formulated WSMO Goal for this request would be as in Figure 2.9.

We assume the existence of the following ontologies in the examples of Figure 2.8 and Figure 2.9:

- the `StockMarketOntology` describing necessary financial and stock market-related concepts and instances;
- the `StockMarketQoSOntology` the QoS domain ontology describing the different quality parameters such as: DataFreshness, Availability, ExecutionTime, etc.
• the euStockMarketMainIndexesOntology describing the various ontological concepts for the euStockMarketMainIndexes() service;

• The UserInformationOntology specifying detailed information about the service users, e.g., user location, day and time of using the service, user’s credit card number, etc.

During the QoS-enabled service discovery, the selection and ranking of Web services in terms of QoS will be done based on the following fundamental semantic-matching steps:

• matching the requested quality level in the interface of the WMSO Goal by the user with the quality level claimed by the provider as specified in each interface of its service description, i.e., checking whether QoSConstraint of the goal subsume that of the service interface.

• testing whether the facts provided by the users in the QoSCondition of the Goal’s Interface fulfill the restrictions that the corresponding service stated in its QoSCondition of the service description’s Interface.

2.4.3 An Algebra for Web Service Discovery

Our solution to QoS-enabled Web service discovery (QWSD) extends traditional database query evaluation strategy to cope with semantic matching between a user goal and Web services description. From a logical point of view, QWSD can be represented as composed by a set of well defined operations. As an example, the selection of web services descriptions based solely on non-semantic QoS Parameter’s criteria can be modeled as a restriction operator. More elaborate restrictions use a match operator that may implement the semantic matchmaking approach proposed in D4.8 [17].

In order to introduce the algebraic approach, we begin by specifying the data structure processed by the algebraic operators. As traditionally adopted in the relational data model, we name the data structure consumed and produced by algebraic logical operators as tuple. The tuple data structure abstracts the different content that it may hold in a certain evaluation. Then, operators communicate by exchanging tuples (or block of tuples) in a producer/consumer fashion. Each operator receives during its initialization a schema of the tuple it will be evaluating describing the actual content of each tuple.

In order to specify a query algebra one must identify the type of queries such an algebra should support. We have identified an initial set of query requirements that drove the discovery algebra design:

• restriction on Web services attribute values - this operation ranks web services descriptions according to the evaluation of a non-semantic database predicate on QoS Parameters. Example of its use would be to select WSDs whose QoS parameter NetworkThroughput has a value above a threshold. Valid comparison operators are: <, >, =, <=, >=. WSDs that agree on a non-semantic predicate receive a rank value of 1, otherwise they receive a rank value of 0.

• matching a goal to a Web service description - this operation implements the semantic matchmaking between a goal and a web service. A possible implementation of this operator complies with the semantic matchmaking approach
as defined in D4.8 [17]. WSDs that semantically match a goal are attributed a matching rank value.

- ranking the result - this operation sorts the result set according to a ranking algorithm;
- producing a result - returns a result set to the user.
- detection of dishonest reports - performs the filtering of dishonest QoS reports from the set of collected data.
- prediction of service QoS - predicts the actual QoS values of a certain Web service based on its historical performance data.

In addition, to the logical operators that cover the functionality required for implementing a discovery algebra, the latter introduces operators that abstract from the logical ones issues associated with reading data from a repository or storing data in a queue for data transference. We name those operators after control operators and introduce them in the sequel:

- obtaining data from WSMO repository - models the retrieval of web service descriptions from a repository and their instantiation into a tuple object;
- transferring data from/to different nodes - specifies queuing strategies and data transfer protocols within a distributed environment. Those operators are used to implement data transfer among parallelized match operators.

According to the above requirements list, the following operations, with their signature, are proposed:

- restriction (σ) - σ(IN: WSD, IN: goalInterface, OUT: rankedWSD) - evaluates non-functional matching based on database predicates expressed over QoSParameters' values in the goal interface. WSDs whose QoSParameter values satisfy the database predicates are marked as matched and no further QoS semantic matchmaking is exerted. The restriction operation, differently than is traditionally accepted for its relational model counterpart, does not eliminate tuples that do not agree on its predicates, since this tuples may be further selected by a semantic matchmaking operation. In this case, its rank value is set to zero.

- match (µ) - µ(IN: WSD, IN: goalInterface, IN: ontologyURI, IN: mediatorUri, OUT: rankedWSD) - evaluates a semantic matching between QoSParameters in goal’s Interface with those in a Web service description. The matching follows the proposal in D4.8 [17]. The ontology URI points to the ontology to be used as the base terminology for the matching. If necessary, a mediator URI identifies the mediator to be used for capturing mappings from concepts used in the WSD into the goal ontology. The result of the µ operator is a list of web service extended with an associated rank value qualifying the level of matching according to QoS parameters. A non matched WSD receives a zero rank value.

- rank (ρ) - ρ(IN:WSD, IN: rankAlgorithm, OUT: rankedWSD) - evaluates a sort of the input according to the algorithm specified in rankAlgorithm.
• project (\(\pi\)) - \(\pi\)(IN: WSD, OUT: rankedWSD) - returns a result set comprising ordered matched web service descriptions.

• dishonesty detection (\(f\)) - \(f\)(IN: QoSReportSet, OUT: QoSReportSet) - filters the list of dishonest reports from the set of collected data.

• prediction (\(\theta\)) - \(\theta\)(IN: WSD, OUT: predictedQoS) - returns the predicted QoS values of a certain Web service description based on its historical performance data.

In addition to the logical operators set, the following control operators complement the discovery algebra:

• scan - scan(IN: repositoryQuery, OUT: WSD) - accesses a repository containing web service descriptions and produces a set of WSD tuples.

• send - send(IN: blockOfTuples, IN: receiveNode, OUT: blockOfTuples) - this operator mediates the data transference between two distributed execution engines. A typical use of this operator is to send tuples to an execution engine, identified as receiveNode, running one of the parallel instances of the match operator. It manages an internal queue for storing incoming block of tuples, which are served to remote receive operators as requested.

• receive - receive(IN: blockOfTuples, IN: sendNode, OUT: blockOfTuples) - receives a block of tuples from a remote send operator, identified as sendNode.

• split - split(IN: blockOfTuples, IN: listOfSendNode, OUT: blockOfTuples) - the split operator controls the distribution of tuples through the various parallel execution engine instances implementing the \(\mu\) operator. It opens a queue for each send operator associated to a remote execution engine. According to a scheduling policy it chooses the queue that should receive the next block of tuples.

• merge - merge(IN: blockOfTuples, IN: listofReceiveNode, OUT: blockOfTuples) - web service descriptions that semantically match the goal are transferred in blocks of tuples to the projection site. The latter runs a receive operator for each node running an instance of the \(\mu\) operator. Merge combines all this incoming tuples into a single dataflow.

Figure 2.10 depicts an example of a discover execution plan expressed using some of the algebra operations above. In this figure, nodes represent operations and directed edges model dataflow. A square surrounding nodes represent the fact that the matching operation is evaluated in different machines.

The following properties are observed:

• interface - all operators implement an iterator type interface (open(), getnext(), close()). This allows for a simple communication protocol between any two operators in the operator tree.

• tuple or block - operators receive either a tuple or a block data structure. Internal tuple structure is known to the operator during its open. This strategy allows for composing algebra operators independently of their logical input/output;
• commutativity - the restriction and match operators are commutative (i.e. \( \sigma(\mu) \equiv \mu(\sigma) \)). The result of a \( \sigma(\mu) \) exhibits ranked wsd with rank value set by one of the operators, which is the same if one evaluated \( \mu(\sigma) \). For WSDs that would match in both semantic and non-semantic matches, we should guarantee that the rank value is the same. This is particularly interesting considering restrictions on the QoS attribute values. The commutative property allows an execution strategy to push-down a restriction below a match predicate. In the case where WSMO Repository implements restrictions, QoS attribute values evaluation is proceeded by the Repository itself;

2.5 Detail Description of the Discovery Component and Implementation Issues

In this part we will briefly describe the functionalities of the discovery component to be implemented in prototype 1 and prototype 2 of our work-package.

2.5.1 The QoS-enabled Service Discovery Process

QoS-enabled service discovery is defined as the process of adding QoS concerns into the Web service discovery process. In this work, the discovery process is extended to cope with QoS by introducing the following operations:

1. the syntactic based QoS-property predicates evaluation;
2. the semantic QoS-enabled discovery;
3. the semantic functionality-based discovery;
4. the ranking of services based on QoS criteria.

This section briefly introduces the main functionality of the above operations within the WSD discovery context based on the WSMO extension proposed in Section 2.4.2.

As it has been presented in Section 2.4.3, our approach models the discovery process as a query execution problem, in which the discovery operations form an algebra. The latter is composed of logical and physical operations. The set of logical operations implements the semantics of the discovery process and comprises the following operations: restriction, matching and ranking. The restriction operation implements a syntactic based QoS-property predicate evaluation. The match operation implements a semantic based discovery and the rank operation orders selected WSDs. In the following, we detail each one of those in the context of QoS-enabled discovery.

The restriction operation considers WSMO goals and WSDs using the same terms to express QoS properties, according to a common ontology. It is specially interesting in the context of QoS properties, in which users generally adopt a standard terminology. WSD discovery, in this context, concerns finding Web services descriptions based on the evaluation of simple predicates on QoS properties. The discovery task is reduced to a direct comparison between corresponding QoS property values in Goals and WSDs. We name the expression used in the comparison a predicate expression and it is formed by a conjunction of the properties specified...
in the \( \text{hasQoSParameters} \) property, as defined in Section 2.4.2. Taking the example depicted in Section 2.4.2, a WSD would fulfill the goal if the following condition holds: \( \text{freeServiceAvailability}.\text{hasLowerBound} = 0.99 \) and \( \text{userLocation} = \text{loc#Techgate} \) and \( \text{accessDay} \neq \text{gday}('01') \).

As discussed in Section 1.4.3, syntactic evaluation only applies in certain constrained scenarios. In more general ones, terms used in QoS properties in the WSMO Goals may not syntactically match with those in the WSDs. In those cases, a more general semantic match making approach is implemented by the \textit{match} operator. We adopt an ontology-based matchmaking approach in which QoS terms in Goals and WSDs are taken as concepts described in a shared QoS ontology. A concept in a goal is said to match with another one in a WSD if a third concept, representing the intersection between the former two ones, is satisfiable by reasoning using the shared ontology \([17, 7]\). The semantic match making would, in opposition to the syntactic evaluation, consider the disjunction of \( \text{QoSParameters} \). For each \( \text{QoSParameter} \) in the goal, a satisfiability test is evaluated against each \( \text{QoSParameter} \) in a WSD, leading to a match whenever the test is satisfiable. Once all the QoS properties have been evaluated a rank value qualifies the WSD in respect to the goal based on the matching concepts.

The execution of QoS semantic matchmaking operators may take a long time, as a function of the number of concepts in goal to be evaluated and the number of WSDs to be checked. In order to reduce the overall elapsed-time, the matching process is parallelized through a grid of nodes. The main parallel strategy is to continuously feed nodes running the \textit{match} operator with WSDs to be matched against the QoS properties extracted from the goal.

The QoS-enabled discovery component communicates with a semantic matchmaker in order to process the \textit{(functionality-based service discovery)} step. The functionality-based service discovery component does the semantic reasoning among the list of services identifying those that satisfy functional requirements. This step basically performs the semantic matchmaking between functional description of each Web service (via the \( \text{hasCapability} \) declaration) and the \( \text{requestsCapability} \) in the submitted WSMO Goal. This phase will return a list of services ranked by the functionality satisfactory level and may require an external data mediation service as described in Section 2.3.1.

Once WSDs have been selected by one or both of QoS matching process and the functional-based service discovery process, the results are order by the \textit{ranking criteria} information provided by the ranking ontology parameter in the \texttt{discover()} API. The final output comprises a list of Web services fulfilling some quality requirements of user, ordered by their associated rank values.

### 2.5.2 QoS Trust and Reputation Management Model

In order to accurately perform the selection and ranking of resulting Web services, we collect user reports on QoS of all services over time and compute their predicted quality. This prediction takes into consideration trust and reputation issues and mainly based on the difference between the promised and the actual delivered quality of the Web services. Since this prediction step would be computationally-expensive but not a time-intensive process, it is performed off-line on a periodical basis so as not to affect system performance and to reflect changes in the environment.
In our framework, the trust and reputation management process is also defined as algebraic operators which could be implemented with different algorithms (as the matchmaking and ranking of services). Other models of trust and reputation management thus could be easily implemented and plugged into the system.

The current trust and reputation model as described in this section is specially designed for the Semantic Web services application scenario and works as the followings.

The step of collecting QoS reports could be done automatically or manually depending on the configuration of the service users themselves. The evaluation of QoS reports by our discovery component has to account for malicious reporting and collusive cheating of users to get a correct view of the QoS properties of a service. Additionally, we also allow trusted agents in the model to provide QoS monitoring for certain services in the system. Their reports are combined with normal user reports to fine-tune the actual QoS characteristics of a service. We describe the first version of our QoS-based service selection and ranking process as well as the details of the trust and management approach in [40]. Briefly speaking, we develop this selection algorithm with two assumptions. Firstly, it is grounded on the probabilistic behavior of services and users. This implies that the differences between the real quality conformance which users obtained and the QoS values they report follow certain probabilistic distributions. These differences vary depending on whether a user is honest or cheating as well as the level of changes in her behavior. Secondly, we presume that there exists a few trusted third-parties. These well-known trusted agents always produce credible QoS reports and are used as a trustworthy information sources to evaluate the behaviors of the other users. In reality, companies managing the service searching engine (the discovery component) can deploy special applications themselves to obtain their own experience on QoS of some specific Web services. Alternatively, they can also hire third party companies to do the QoS monitoring tasks for them. The main idea of this algorithm is to use trust-distrust propagation and clustering method to estimate the credibility of the QoS reports from many different users. Relying on the set of honest reports discovered by this algorithm, we try to predict approximately the future QoS parameter values of related Web Services using regression method. The predicted values of various quality parameters of the services in the systems will be stored in the QoS Data Repository of the discovery component as described in Section 2.2. The QoS-enabled service discovery process then uses the predicted information from this data store to perform the selection and ranking of the services based on the QoS criteria specified by the user.

The above approach is employed by our discovery component for the following reasons: the strong validity of the assumption on the existence of trusted rating agencies, e.g., AlertSite [1], Dot-Com Monitor [2], and Empirix [3], etc.; the effectiveness of the method [40], and the ability of such a model to keep the related partners have incentives to cooperate with each other in the long term [20].

2.5.3 WSMO Repository Implementation

The QoS enabled Web service discovery, as depicted in 2.1, retrieves a set of candidate Web Services descriptions from a WSMO repository. In this section, a Repository supporting the storage and retrieval of WSMO ontologies and Web service descriptions is presented. The repository design complies to three main requirements: (1) It conforms to WSMORegistry interface, defined in WSMX infomodel; (2) it supports the storage
and retrieval of WSMO Web service descriptions and ontologies; and (3) it supports queries with restrictions over entities attribute values.

The first requirement is achieved by offering a facade class that conforms to the WSMORegistry interface. The storage and retrieval of WSDs and ontologies require these entities to be serialized in a XML file. The repository stores the complete XML file in its native format into a RDBMS. Thirdly, the repository is implemented on top of a full fledged DBMS, which enables ontology entities to be retrieved based on predicates over attribute values. A common scenario that illustrates its usefulness is a restriction over non-functional property values. A query to retrieve a WSMO entity is formulated using a variation of the XPATH language adapted to closely reflect the repository model. As an example, to obtain concepts defined in an ontology named location, a user would specify: "/ontology[name="location"]/Entity[type="concept"]". As a second example, to obtain Web service descriptions with a MeanValue for the QoS parameter Throughput greater than 3.0, the following query would be specified: 

```
projection = /webservice/Entity[type = "WebService"]
Restriction = /webservice/Entity[type = "WebService”]/Interface//metainfo
            /property[name = "Throughput”]/value/ > 3
```

A complete language definition can be found in [36].

Having the query specified over a meta-model completely hides the storage model from the user. This means that the proposed repository is completely independent of the storage strategy implemented. Queries submitted to the repository are automatically re-written to the target physical storage, according to specified mappings. In the current specification, the physical layer adopts the XML data model. Native XML storage is in these days supported by the main DBMS providers, such as Oracle10g or DB2, leading to its easy integration to the remaining relational data. Storing ontology data in XML considerably reduces the complexity of this task as data is kept in its original serialized format. In this context, mappings are specified that associate meta-model entities to paths of the XML document representing stored entities, such as ontology; ontology entity; Web service description and its components; mediators, etc. A mapping must be specified for each new XML schema encoding of an ontology or Web service description. The present solution will provide a full query re-writing implementation for the chosen native XML storage solution. Different solutions, other native XML storage or a different data model require the specialization of the MapStorage class.

An example of a mapping document is presented in [36]. Storing a WSMO ontology and Web service description is done by simply using the method public void store (WSMLDocument wsmlMessage) passing as input parameter the WSMO entity as the content of WSMLDocument. If the stored document schema complies to an existing mapping definition than the document is available for querying as soon as it is entered into the Repository.

2.5.4 Detailed Designs of the QoS Discovery Component

Due to space constraints we only provide the central five (5) diagrams out of 23 UML design diagrams of the QoS service discovery component in the description to follow and briefly discuss them. All other diagrams can be found on the BSCW server at https://bscw.dip.deri.ie/bscw/bscw.cgi/d63750/UMLDesign.zip or at http://lsirpeople.epfl.ch/lhvudownload/qosdisc/doc/UMLDesign.zip.
The diagram in Figure 2.11 depicts the main functionalities provided by the QoS discovery component. The main actors for the system are the user, the service provider, the system administrator and certain trusted 3rd-party agents. The main use cases are AdvancedDiscovery and DiscoveryService, corresponding to the two methods of service discovery with and without a ranking ontology as specified by the component API in section 2.4.

Figure 2.12 illustrates the different states of the QoS discovery system throughout its lifetime.

Figure 2.13 is the design of major operators of the discovery algebra as specified in Section 2.4.3.

The discovery operations take places as in Figure 2.14 and Figure 2.15, therein the class DiscoveryQueryManagerImpl implements the functionalities of the query processing system CoDIM-S-G [31] with the additional discovery algebraic operators specified in Section 2.4.3.

2.5.5 Task List

The implementation of the a full-fledged QoS-enabled service discovery component consists of the following tasks:

1. Develop sample ontologies: QoS ontologies for the eBanking, the B2B Telecomm case study, as well as example ranking ontologies.

2. Working with related partners to update the WSMO conceptual model and associated tools.

3. Implement the listed subcomponents: Query Analyzer, Query Optimizer and Execution Manager, Semantic Query Engine.

4. Implement a semantic matchmaker for QoS ontological concepts.

5. Implement the QoS Reputation Mechanism.

6. Design the QoS Data Repository and Internal WSMO Service Repository.

7. Integrate with the eBanking and the B2B Telecomm use case partners and (optionally) with the mediation service.
Representation of QoS information in a WSMO Web service description

namespace {
  _"http://example.org/euStockMarketMainIndexes.wsml#",
  dc _"http://purl.org/dc/elements/1.1#",
  wsm _"http://www.wsmo.org/wsm1/wsm1-syntax#",
  loc _"http://www.example.org/LocationOntology.wsml#",
  qos _"http://www.example.org/StockMarketQoSOntology.wsml#",
  userInfo _"http://www.example.org/UserInformationOntology.wsml#"
}

webService euStockMarketMainIndexes
importsOntology {"_http://example.org/StockMarketOntology.wsml"
}
capability euStockMarketMainIndexesCapability
interface euStockMarketMainIndexesInterface_FreeService
importsOntology
  {"_http://www.example.org/StockMarketQoSOntology.wsml",
   _"http://www.example.org/UserInformationOntology.wsml",
   _"http://www.example.org/LocationOntology.wsml"
}
nonFunctionalProperties
dc#description hasValue "QoS properties of free stock market information web service."
QoSParameter availabilityLevelForFreeService
QoSConstraint definedBy
  ?freeServiceAvailability[
    hasLowerBound hasValue 0.99,
    hasUpperBound hasValue 1.0
  ] memberOf qos#Availability
QoSCondition definedBy
  ?userData[
    userLocation hasValue loc#Techgate,
    accessDay hasValue ?accessDay,
  ] memberOf userInfo#userInformation
and ?accessDay != _gday("01").
endNonFunctionalProperties

choreography euStockMarketMainIndexesChoFree
orchestration euStockMarketMainIndexesOrchFree
interface euStockMarketMainIndexesInterface_PaidService
importsOntology
  {"_http://www.example.org/StockMarketQoSOntology.wsml",
   _"http://www.example.org/UserInformationOntology.wsml",
   _"http://www.example.org/LocationOntology.wsml"
}
nonFunctionalProperties
dc#description hasValue "QoS properties of paid stock market information service."
QoSParameter availabilityLevelForPaidService
QoSConstraint definedBy
  ?paidServiceAvailability[
    hasLowerBound hasValue 0.999999
    hasUpperBound hasValue 1.0
  ] memberOf qos#Availability
QoSCondition definedBy
  ?userData[
    hasCreditCardNumber hasValue ?credNo
  ] memberOf userInfo#userInformation
and userInfo#validCreditCard(?credNo).
QoSParameter dataFreshnessForPaidService
QoSConstraint definedBy
  ?paidServiceDataPressNess[
    hasLowerBound hasValue 0
    hasUpperBound hasValue 10
    hasMeasurementUnit hasValue qos#minute
  ] memberOf qos#DataFreshness.
QoSCondition definedBy true.
endNonFunctionalProperties

choreography euStockMarketMainIndexesChoPaid
orchestration euStockMarketMainIndexesOrchPaid

Figure 2.8: Representation of QoS information in a WSMO Web service
Representation of QoS information in a WSMO Goal

namespace _"http://example.org/euStockMarketMainIndexesGoal.wsml#",  
dc _"http://purl.org/dc/elements/1.1#",  
wsml _"http://www.wsmo.org/wsml/wsml-syntax#", 
loc _"http://www.example.org/LocationOntology.wsml#",  
qos _"http://www.example.org/StockMarketQoSOntology.wsml#",  
userInfo _"http://www.example.org/UserInformationOntology.wsml#")

goal euStockMarketMainIndexesGoal
importsOntology _"http://example.org/StockMarketOntology"
capability euStockMarketMainIndexesRequestedCapability

interface euStockMarketMainIndexesRequestedInterface
importsOntology  
{ _"http://www.example.org/StockMarketQoSOntology",  
_"http://www.example.org/UserInformationOntology",  
_"http://www.example.org/LocationOntology.wsml")
nonFunctionalProperties
dc#description hasValue "Requested QoS properties of the stock market information web service."
QoSParameter requestedDataFreshness
QoSConstraint definedBy
?requestedDataFreshnessInstance[  
hasUpperBound hasValue ?max  
hasMeasurementUnit  
hasValue qos#minute
] memberOf qos#DataFreshness
and (?max <= 15.0).
QoSCondition definedBy
?userData[  
userLocation hasValue loc#ViennaIntAirport,  
] memberOf userInfo#userInformation.
endNonFunctionalProperties

choreography euStockMarketMainIndexesChoPaid
orchestration euStockMarketMainIndexesOrchPaid

Figure 2.9: Representation of QoS information in a WSMO Goal
Figure 2.10: Example of a discovery execution plan
The user could be normal people, computer experts, or even autonomous agents acting on behalf of users.

Figure 2.11: Use case diagram
SYSTEM STATE DIAGRAM (QoSServiceDiscoverySystem class)

Figure 2.12: System state diagram

Figure 2.13: Operator design class diagram
This advanced discovery activity is similar to discovery, but with the following enhancements:

- The Request is generated with further information from ranking ontologies.
- Mediation services may be used during the CoDIMS-G:QEEF execution process.
- This is expected to be implemented in Prototype 2 (M36).
3 Example Scenarios

To demonstrate and ensure the applicability of DIP technologies in real-world scenarios, it is expected that every DIP technology to be grounded into a use case. This allows the technology providers to verify their technologies against the requirements of the use cases and enables the use cases to evaluate the provided DIP technologies in their domain of interest. Moreover, this assures the correct assessment of the use case requirements and simplifies the incorporation of novel DIP technologies into the use cases.

In this section, we are going to give some concrete examples in which the application of our QoS-enabled service discovery component is highly required. Towards the above overall goal of the DIP project, we develop our example scenarios based on the coordination with the e-Banking and (WP10) the B2B Telecomm (WP8) case studies, which is reflected in Section 3.1 and Section 3.2, respectively. Section 3.3 and Section 3.4 demonstrate two other important applications of our QoS-aware service discovery in the area of e-Business and e-Science, thus clarifying the realistic applicability of the technologies developed by our group\(^1\).

3.1 Searching for the best Stock Information Service

A service provider, Bankinter, offers several services to consult different kinds of Stock Market information (news, charts, index variations, and stock prices), services to sell and buy stocks, services to send alerts and others, etc.

Bankinter publishes the semantic descriptions of its services into a WSMO registry. These descriptions include the information of different QoS parameters and the corresponding conditions for achieving these levels of quality. The most relevant QoS parameters are:

- the freshness of the provided information from the stock market, e.g., the price of an index should reflect its real value with a maximum delay of 20 minutes;
- the breadth of stock market coverage offered by the service;
- the average response time of the service;
- the average execution time of the service;
- the number of requests per minute that the service can handle;

Note that the first two QoS attributes are application-dependent quality properties, which have special meanings in the Stock Market application domain, whereas the last three QoS parameters are generic software-level performance attributes, which are also important for those users who want to integrate the provided Stock Market information services into their own applications, e.g., a service-based financial analysis software.

The prerequisites for achieving a certain QoS level could be:

\(^1\)One can find the implementation of these examples, which is based on the current WSMO conceptual model, from the QoS discovery component download page (http://lsirpeople.epfl.ch/lhvu/download/qosdisc/). The changes of those implementation to the extended WSMO model, once this is available, is trivial.
• the subscription price associated to each information quality level, e.g., free or non-free;

• the minimal speed of the Internet network connection of the user;

• the location of the user, i.e., access to some services through mobile phone connections are restricted to certain zones, regions, countries, etc.;

• the additional equipments and software needed in the client machine;

The Stock Market application helps users to find and use Web Services that satisfy both functional and quality requirements. For example, one could require a Stock Market Information Web service to exhibit the following quality criteria:

• a freshness rate for the provided information varying from 20 to 21 minutes;

• an average service response time between 1.5 to 2.0 seconds;

• a service processing capacity of at least 50,000 requests per minute;

The user also provides extra information concerning her accepted conditions and other environmental factors, which would be used as prerequisites when searching for a service that suits her QoS requirements:

• the maximum subscription price she is willing to pay, e.g., 5.0 Euros per hour;

• her Internet access, e.g., a wireless-based 10Mbps connection;

In this case, the task of searching and selecting the best services for this application is QoS-sensitive and thus could be handled by our QoS-enabled service discovery component. Specifically, the Stock Market application will formulate a WSMO Goal which consists of both functional and QoS requirements of the user and call the unified API of the discovery component to get a list of matched services. This call to the unified service discovery API will be sent to both the functionality-based and QoS-enabled service discovery component. The results of the discovery process are a list of matched Web service descriptions, ranked by their matching levels to the requirements of the user. The user will choose one Web service that suits her needs, after which the Stock Market application will help her to invoke that chosen service.

3.2 Selecting the best Products/Services

The main goal of the BTs Workflow Management System (WFMS) is to enable the automatic management of various business actives of BT, such as the interactions with its business partners, the management of product catalogues, the placement of orders to suppliers, the advertisement and selling of products and services. The interaction of this WFMS with other business partners is based on the execution of various Web services. For example, a supplier would provide Web services for browsing and ordering its products. Furthermore, these services allow BT to use or resell various products directly via the service interface in certain ways (in case the products are online services

\(^2\)http://www.bt.com
such as data backup, hosting, housing services, etc.). The use of Semantic Web service technologies enables the design and implementation of an highly adaptive WFMS where the choice of component services provided by various business partners could be done on-the-fly for different products, according to BTs actual business needs.

As the automatic operation of the WFMS strongly relies on the smooth execution of the component services offered by the various partners of BT, QoS is an important criterion while selecting and integrating these services into the system. Example QoS requirements for WFMS component services are:

- the availability of the service and its meantime between failures;
- the average response time of the service;
- the number of requests per minute this service can handle;
- the security level of the service;
- the execution time of the service;
- the throughput of the services, i.e., the number of requests processed successfully per time unit;

Note that in this case, the application-level and software-level QoS parameters of the component services are the same as these quality factors mean the quality of the BT’s WFMS, which is a software system itself. Also, the provider may not always explicitly make their claims of such QoS attributes in the associated service descriptions. In such a scenario, a WFMS (as user of the services) could monitor, evaluate and submit collected QoS information to the QoS-enabled service discovery component. Obtained statistics are used to enhance the selection of the best services based on their QoS compliance to be integrated in the WFMS.

Alternatively, for certain products which are offered as online services, a supplier usually includes in the associated descriptions the functionality and the QoS parameters (or more generally, non-functional properties) of the product services in its list of product offers, which is to be searched by the BT’s WFMS. Thus, the WFMS can also call the QoS-enabled service discovery component to do the QoS-based matchmaking and get the list of matched products for its own use.

An example of such a product is a professional data backup service for companies and individuals. The QoS of such online backup service consists of the following parameters:

- the availability of the product service, e.g., it should be available for the whole week and the mean time between failures should be less than 30 s;
• the response time of the service, e.g., should be less than 2 s;
• the maximum throughput of the service, e.g., should be capable of processing at least 5,000 requests per second;
• the capacity of the data storage, e.g., greater than 3GB;
• the upload speed (to backup data), e.g., should be at least 20 KBps;
• the download speed (to restore data), e.g., should be at least 60 KBps;
• the price of the service, e.g., bellow 5.0 Euros per each GB of storage per month;
• the security level of the product service, e.g., data in the server should also encrypted with 256-bit DES standard.

In this case, the corresponding suppliers will describe these product services semantically using the WSMO model and publish them in their own WSMO registries. The description of the product services can also include condition to achieve the necessary QoS level. Examples of such conditions are:

• the subscription price to use the product service;
• the minimal speed of the network connection;
• the equipment and software needed in the client machine;

To perform the task of searching and selecting the best product services, the WFMS will formulate a WSMO Goal which consists of both functional and QoS requirements for the needed product and call the unified API of the discovery component to get a list of matched product services. This call to the unified discovery API will be sent to the functionality-based and the QoS-enabled service discovery component. The result of the discovery process is a list of matched Web service descriptions, ranked by the matching level to user requirements.

Additionally, the WFMS (or the catalog management system in the second case) may also monitor the various QoS parameters of the invoked Web services (or the advertised products) and report the collected values to the QoS-enabled discovery component via other APIs. These reports will help the discovery component to evaluate the real quality level of the associated services and be able to provide the more accurate values for answering future requests from the WFMS. Suppliers and other partners of BT can also query the QoS-enabled discovery component directly to learn about the actual performance of its services/product services and improves them if necessary.

3.3 Searching for the suitable Internet Service Provider

Another candidate application of the QoS-enabled discovery component is the Virtual Internet Service Provider (VISP) use case, which was also part of the work of WP8.4. In a nutshell, the idea behind the VISP business model is that Internet Service

4This case study has been integrated into the B2B Telecom Use Case of the DIP project recently.
Providers (ISPs) describe their services as Semantic Web services, including QoS such as availability, acceptable response time, throughput, etc., and a company interested in providing Internet access, i.e., becoming a VISP, can look for its desired combination of services taking into account its QoS and budgeting requirements, negotiate and provide its service. The VISP then uses this service for its own applications, e.g. creating a new Internet service product for end-users. At the moment this business model exists, but is done completely manually [19, 15].

Many (virtual) products offered by the VISP are QoS-sensitive and for this reason, the VISP would strongly need to search a QoS-enabled discovery component to search for the most suitable Web services. According to [15], some of these products are:

- a hosting service (FTP, WWW);
- a housing service for application of the users;
- a portal service for providing online-game access, communication services, etc.

Let us take a typical example where the VISP would like to create a new virtual product which offers low-price and high-quality hosting service for its users. For such a situation the VISP needs to search the various service advertisements of different providers for the most suitable data storage Web services to be used in its final product. The quality requirements for this service include, for example:

- the service should be available for the whole week and the mean time between failures should be less than 20 s;
- the service should be capable of processing at least 5,000 requests per second;
- the capacity of the data storage is greater than 5GB for each user account;
- the upload speed should be at least 10 KBps;
- the download speed should be at least 20 KBps;
- the price of the service is below 0.5 Euros per each GB of storage;

Those above QoS capabilities could be described using a QoS domain ontology. The ISPs can provide different Web services that fulfill those above requirements of the VISP at different levels and with various pricing models. A lot of service descriptions with different quality levels are advertised in the WSMO service registry during the service publishing phase. Moreover, dishonest providers could claim arbitrary QoS properties to attract interested parties. The standard way to prevent this is to allow users of the service to evaluate a service and provide feedbacks. However, the feedback mechanism has to ensure that false ratings, for example, badmouthing about a competitor’s service or pushing own rating level by fake reports or collusion with other malicious parties, can be detected and dealt with. Therefore, our discovery component also takes into account not only the functional suitability of services but also its prospective quality offered to end-users regarding to the trustworthiness of both providers and consumer reports.

The results of the search process is a list of service descriptions and their corresponding interfaces. This list is further ranked according to the suitability of the
interface of each service with the request of the VISP agent. In the service definition step, the VISP selects the most appropriate service, usually one of top-K services in the result list, performs other compatibility checking, negotiation, etc., and finally executes the Web service. The perceived quality levels, which are estimated by the VISP platform or by the end-users are then reported to the QoS-enabled discovery component via the discovery component API.

3.4 Searching for the best Computational Service

In the area of e-Science, computational-intensive Web services are frequently used as components for building scientific applications, e.g., scientific visualization, weather forecasting and climate studies, system modeling and simulation, tele-operation supported software, etc. In such scenarios, the technologies developed in our QoS-enabled service discovery component could be used to search and select the best Web services in terms of QoS criteria for the user’s application. For example, a scientist would like to search for a Web service which is capable of solving a linear equation system with the following QoS requirements:

- the mean-time between failures of the service should be less than 0.5 s;
- the response time of the service should be less than 0.05 ms.
- given an input size of 500,000 equations, the system should be able to return the results as latest as after 5 hours of execution with the accuracy up to 6 digits.
- the service can handle at least 5,000 requests per minute.

This usage scenario proves that the capabilities offered by our QoS-enabled discovery service are more powerful and offer functionalities even beyond the requirements of the DIP case studies. Along with the recent alignment of work between the SDK cluster and the Adaptive Grid Service initiative\(^5\), this contribution of our component becomes more and more important to the DIP project.

4 Proposed Extensions of the WSMO Model

In order to develop a full-fledged QoS-enabled service discovery component, along with the extension of the WSMO conceptual model as described in Figure 2.7, the following changes to syntax of the WMML language are expected (Figure 4.1). Other related tools such as the WSML validator, wsmo4j API, etc., should also be updated to reflect these modifications as well.

The proposed extension for the syntax of the WSML language

```
nfp = 'nfp' nfpContent 'endnfp' | 'nonFunctionalProperties' nfpContent 'endNonFunctionalProperties'
nfpContent = attributevalue* qosQoSParameterDefinition*
qosQoSParameterDefinition = 'QoSParameter' id? sharedvardef? qosConstraintDef qosConditionDef?
qosConstraintDef = 'QoSConstraint' axiomdefinition
qosConditionDef = 'QoSCondition' axiomdefinition
```

Figure 4.1: The proposed extension for the syntax of the WSML language

Regarding the wsmo4j API [5], we would need the following extensions:

- A definition of the QoSParameter class as specified in the extension to the WSMO conceptual model. We need the constructor, the get and set methods for the related variable members of this new class.
- The additional methods for setting and retrieving the newly added QoSParameter objects to the non-functional properties of the Entity class.

Figure 4.2 elaborates these newly required APIs for WSMO4J:

The proposed extension for the WSMO4J API

```
Class org.wsmo.common.QoSParameter
    org.omwg.ontology.QoSParameter QoSParameter(org.omwg.ontology.Axiom qosConstraint, org.omwg.ontology.Axiom qosCondition);

    org.omwg.ontology.Axiom getQoSConstraint();
    void setQoSConstraint(org.omwg.ontology.Axiom qosConstraint);

    org.omwg.ontology.Axiom getQoSCondition();
    void setQoSCondition(org.omwg.ontology.Axiom qosCondition);

Class org.wsmo.common.Entity
    java.util.Set listQoSParameters();
    void addQoSParameter(IRI key, org.omwg.ontology.QoSParameter qosParameter);
    void removeQoSParameter(IRI key, org.omwg.ontology.QoSParameter qosParameter);
```

Figure 4.2: The proposed extension for the WSMO4J API
5 Conclusion

DIP’s vision is the further development, combination and enhancement of the Semantic Web and Web service technologies to produce a new technology infrastructure. In order to realize this challenge, DIP has to provide new languages, tools and architectures that extend the current technologies with means for leveraging novel semantic ideas into large-scale B2B environments.

In this deliverable we have contributed with an important technological piece to the DIP’s objective. We identified, analyzed and proposed a new approach for extending standard functionality-based Web service discovery with discovery based on quality of services criteria. This deliverable presents the requirements for QoS information associated with Web services and proposes an extension to the WSMO ontology language to accurately express QoS information. The Web service discovery approach is therein extended with a new component that deals with discovery based on QoS criteria. Our extension is in accordance with the wsmo4j API, the WSMX architecture and the DIP’s functionality-based service discovery component. A fundamental aspect of QoS is to provide a means for measuring the trustworthiness of published QoS information. Thus we propose a new trust model that reduces the impact of cheating on advertised QoS information.

The enhanced Web service description offers a substantial advantage for B2B applications as it has been observed by the use case partners, such as the eBanking (WP10) and the B2B Telecom case study (WP8), that QoS-based service discovery is highly relevant in real-world scenarios. QoS-based discovery allows the user to access the quality of services that different providers advertise within an integrated WSMX environment.
REFERENCES


