WISEMOD
A Framework that Combines Workflow, Ontology and Semantic Web Service Technologies

Proof of Concept

Even André Fiskvik
even@lynweb.no

Jørgen Rygh
jorgen.rygh@tihlde.org

TDT4900 Master Thesis, Technology Programme
June 7, 2006

Teaching supervisor:
John Krogstie
john.krogstie@idi.ntnu.no

Co-supervisors:
Rune Molden Terje Wahl
rune.molden@idi.ntnu.no terje.wahl@idi.ntnu.no

NTNU
Innovation and Creativity

Norwegian University of Science and Technology, Department of Computer and Information Science
Abstract

This project builds upon the results of a prestudy performed by the same authors in the latter part of 2005. Both this project and the prestudy was performed under the ongoing WISEMOD effort at the Norwegian University of Science and Technology. The prestudy showed that Semantic Web technologies could be used to automate many scenarios where information is processed or retrieved using traditional Web Services. To enable the creation of clients supporting Semantic Web Services, the need for a solid back-end system was clear – a Semantic Web Service Gateway (SWSG) was needed. The specification and implementation of a prototype of this back-end system is the topic of this report.

The SWSG prototype developed in this project provides a reference implementation of a back-end system that provide Semantic Web Services to end-user clients and Workflow Management Systems. The prototype was built on an architecture that enables the system to be extendable, modifiable, testable and in general very flexible for future usage and research. The prototype was largely based on off-the-shelf third party components, so much of the job was related to figuring out how these components could be used and how they could work together.

During the project we have refined the requirements presented in the prestudy report, and used these to define an architecture for the SWSG system. We have also implemented a runnable proof-of-concept prototype based on this architecture. While implementing the prototype we have investigated some of the many available tools and frameworks for the Semantic Web. Our experiences with these tools are included in this report.

The prototype that has been created during this project shows how Semantic Web Services can be provided to clients that not necessarily themselves are semantic aware. The prototype system can be used for the creation of intelligent end-user applications, or it can be used for supporting dynamic business processes through Workflow Management Systems. The prototype can also be used for testing new components for the Semantic Web as they become available, since it is built on a highly extendable and flexible architectural framework.
Preface

This project is a follow up to the prestudy [16] conducted on Workflow, Ontology and Semantic Web Service technologies by the authors of this report in 2005. The project was executed from January 2006 to June the same year and is the concluding master thesis for the TDT4900\textsuperscript{1} Technology Programme at the Norwegian University of Science and Technology\textsuperscript{2}. The project was assigned by the Department of Computer and Information Science\textsuperscript{3} in cooperation with the ongoing WISEMOD project [49].

Acknowledgements

We would like to thank Njål Arne Gjermundshaug for sharing his experience with various OWL-S tools and components. We would also like to thank our teaching supervisors Rune Molden and Terje Wahl for their guidance and sharing of expertise during this project.

Trondheim, June 7, 2006.

Even André Fiskvik

Jørgen Rygh

\textsuperscript{1}http://www.idi.ntnu.no/undervisning/kursinfo.php?emnekode=tdt4900
\textsuperscript{2}http://www.ntnu.no/
\textsuperscript{3}http://www.idi.ntnu.no/
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 Future Work</td>
<td>97</td>
</tr>
<tr>
<td>18 Conclusions</td>
<td>101</td>
</tr>
<tr>
<td>Bibliography</td>
<td>102</td>
</tr>
<tr>
<td>Appendix</td>
<td>107</td>
</tr>
<tr>
<td>A Glossary</td>
<td>109</td>
</tr>
<tr>
<td>B Maven2 Command Reference</td>
<td>111</td>
</tr>
<tr>
<td>C File Attachements</td>
<td>113</td>
</tr>
<tr>
<td>D Source Code</td>
<td>115</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Elements in the Travel Domain.</td>
<td>19</td>
</tr>
<tr>
<td>4.2</td>
<td>Elements in the Supply Chain Domain.</td>
<td>20</td>
</tr>
<tr>
<td>4.3</td>
<td>Elements in the Online Shopping Domain.</td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>Elements in the Scheduling Domain.</td>
<td>22</td>
</tr>
<tr>
<td>4.5</td>
<td>Elements in the Information Searching Domain.</td>
<td>23</td>
</tr>
<tr>
<td>7.1</td>
<td>Limitations of the prototype implementation of the Client Access Service</td>
<td>37</td>
</tr>
<tr>
<td>7.2</td>
<td>Limitations of the prototype implementation of the Façade Service</td>
<td>39</td>
</tr>
<tr>
<td>7.3</td>
<td>Limitations of the prototype implementation of the Matchmaker Service</td>
<td>40</td>
</tr>
<tr>
<td>7.4</td>
<td>Limitations of the prototype implementation of the Selection Service</td>
<td>41</td>
</tr>
<tr>
<td>7.5</td>
<td>Limitations of the prototype implementation of the Invocation Service</td>
<td>42</td>
</tr>
<tr>
<td>B.1</td>
<td>Maven2 commands for handling the project</td>
<td>111</td>
</tr>
</tbody>
</table>
List of Figures

2.1 The Scrum Process Model, figure used with permission from Mountain Goat Software .................................................. 8
4.1 The external architecture of the proposed system. .................. 17
4.2 UML Domain Model showing the concepts described in the scenarios. 18
6.1 Service Layers in the system ........................................... 34
6.2 System sketch from the prestudy [16] ................................. 36
8.1 Workflow Execution ....................................................... 46
8.2 User’s workflow interacting with the RE .............................. 47
8.3 Graphical User Interface mockup for the main window. ........... 49
8.4 Graphical User Interface mockup for ontology concept selection. 50
9.1 Overview of the various modules in the Spring Application Framework, figure from the Spring website [45] ......................... 57
10.1 UML Class diagram showing the Shared Domain Model ........... 60
10.2 UML Class diagram showing the Registry Service .................. 62
11.1 UML Class diagram showing the Matchmaker package .......... 66
11.2 UML Sequence Diagram showing of the Matchmaker Service .... 67
11.3 UML Class diagram showing the classes in the Selection Service 69
11.4 UML Class diagram showing the Invocation package .............. 72
11.5 UML Sequence Diagram showing of the Invocation Service ...... 73
12.1 UML Class diagram showing the Façade Service .................. 76
12.2 UML Sequence diagram showing an automatic process performed with the Façade Service ................................. 77
12.3 UML Sequence diagram showing a manual process performed with the Façade Service ........................................... 78
13.1 UML Class diagram showing Client Access Service ................. 82
Chapter 1

Introduction

This is the final report for the follow up project to the depth study [16] which ran in the second half of 2005, this previous project is henceforth known as the prestudy. The project aims to implement a prototype of the system discussed in the prestudy. The prototype will be built using an architecture that will provide a general framework for creating back-end systems that can enable the use of Semantic Web Services within end-user clients.

1.1 Motivation

The amount of information and services available on the web is growing every day [34]. To utilize all these available resources to their full potential a new breed of tools are needed – tools that are intelligent enough to automate many of the tedious manual operations needed to discover and invoke Web Services. There has been made considerable progress in creating new standards for managing service composition and handling issues such as security and trust in Web Services through the WS-* standards from the Web Service Interoperability Organization [53].

The Semantic Web [5] and Semantic Web Services [30] have received increased attention since Tim Berners-Lee wrote his influential article [5] on the Semantic Web in 2001. Several projects are researching Semantic Web technology, and multiple tools and frameworks based on Semantic Web technology are under development. The prestudy [16] to this project showed that Semantic Web technologies could play a vital part in enabling the creation of this new breed of applications.

The prestudy project also showed that in order to create intelligent clients based on Semantic Web technologies, a back-end system was needed. We have called this back-end system a Semantic Web Service Gateway (SWSG), since it acts as a gateway to Semantic Web Services for traditional client applications. This project focuses on defining and implementing a prototype of a SWSG back-end system.

\footnote{A set of different emerging Web Service standards.}
There are already a few projects that work on creating environments for enabling discovery, composition and invocation of Semantic Web Services, but these environments are currently either unfinished or uncomplete. Also, the existing Semantic Web Service frameworks are created for research purposes and tend to be difficult to use in practice. This has lead to our proposal for a new system that can provide Semantic Web Services to traditional applications with as little effort as possible.

The SWSG system can particularity support the creation of new tools for Knowledge Workers. A knowledge worker, as originally coined by Peter Drucker in his book “Landmarks of Tomorrow” [13], is anyone whose daily work consists of using or developing knowledge or information. The typical tasks of a knowledge worker include; planning, acquiring, searching, analyzing, organizing, storing, programming, distributing, marketing, or otherwise contributing to the transformation and commerce of information. In other words a knowledge worker has great needs for finding and processing information. Today this information and the services that process the information is often available as Web Services found on the Internet or as local services on an Intranet. These services are either manually invoked by the knowledge workers or they are managed through static workflows that represent the business processes. Semantic Web technology can remove much of the manual effort needed to use Web Services and also help create dynamic and self-optimizing workflows.

Semantic Web Services can also be beneficial to Workflow Management Systems. Workflow technologies such as WS-BPEL [9] allows organization to create workflows that implement their business processes as a dataflow where information passes through a series of automated or manual services. Semantic Web Services provided by our SWSG system can allow increased flexibility to such workflows, allowing the creation of truly dynamic workflows. This will be increasingly important as organizations become more and more dynamic, and often alter their business processes to fit changing markets.

1.2 Problem Definition

This project is based on a previous project [16] that studied the technologies available within the Workflow, Ontology and Semantic Web Service fields of research. A large number of available technologies were selected and investigated in order to figure out how they could be combined in a complete framework or environment for Semantic Web Services. Existing similar frameworks have been investigated in order to identify their strengths and weaknesses, and how they potentially could be further improved and / or extended.

The project will develop a prototype of a framework that combines Workflow, Ontology and Semantic Web Service technologies. The framework will provide the infrastructure needed to implement a back-end system to support the next generation of intelligent knowledge worker clients. The need for such a framework was found from the results of the prestudy project [16], before a suitable framework and back-end system for providing Semantic Web Services to clients have been developed, the creation of clients that use
the features of the Semantic Web will be very difficult. Hence this project will not focus on the creation of end-user clients, but focus on the underlying infrastructure in form of a back-end system based on a flexible framework. The system should be based on open standards in order to secure free usage for future industry and research purposes.

The project will continue to refine the elicited requirements for a new system based on the experiences gained from investigating existing projects and technologies in the prestudy [16]. A detailed design and architecture will be developed, and a prototype for this new system will be implemented as a proof of concept. This prototype will be based on using, modifying and/or extending existing systems and components where appropriate. The resulting system will be documented and evaluated in this final report.

The concrete goals of this project can summed up as:

1. Refine the requirements defined in the prestudy report [16] and select which features to focus on during this project.
2. Investigate the possibility of using existing frameworks and tools that will cover the requirements from the previous point.
3. Create an architecture and design for a framework that can be used to implement supporting back-end systems for enabling the use of Semantic Web Services in end-user clients.
4. Implement a proof-of-concept system or prototype using the above mentioned architecture and framework.

1.3 Problem Context

The work documented in this report is related to the Web Information Service Modeling (WISEMOD) project [49], funded by the Research Council of Norway (NFR).

The concrete goals of the WISEMOD project are:

- Develop an integrated approach for ontology and information service engineering including techniques for
  1. Creating web service workflow and orchestration models
  2. Information service provision and consumption
  3. Quality management of the process and product in Web information systems development.
- Develop an evaluation framework for the quality of web services
- Publish the results as scientific articles in journals, and conferences, technical reports, graduate and undergraduate courses, PhD and Diploma Thesis, lectures, international workshop, and seminars more directed towards industry and public organizations that may be later users of the results.

http://www.forskningsradet.no/forport/application?lang=en_UK
1.4 Limitation of Scope

The focus of this project is to create a proof-of-concept for the system described in the prestudy report [16]. Due to the complex nature of the SWSG system proposed in [16] the implementation will be limited to parts of the complete system. The modules are to be implemented will be determined later in this report. This proof-of-concept system will also be based on an extendable architecture, or framework, so that the system can be extended or reimplemented using the framework as a base.

The prototype that will be developed can be called a Semantic Web Service Gateway (SWSG), a system that can provide Semantic Web Services for non-semantic clients. This means that the prototype developed in this project will be the back-end part of the system discussed in the prestudy [16]. I.e. the system developed here will not be an end-user system, or a knowledge worker client, but rather a system or gateway that enables the creation of end-user applications.

It is important to stress that existing solutions such as tools and libraries should be used whenever available. The projects and technologies investigated in [16] must be further investigated in order to decide whether or not they may be using in our prototype implementation.
1.5 Reader’s Guide

This section describes the structure and content of this report.

Chapter 1 Introduction

The introduction chapter presents the motivation, goals and limitations of the project.

Chapter 2 Method

The method chapter describes the methodologies and processes used in the different phases of the project and provides a short description of the tools used.

Part I Requirements

Chapter 3 Introduction

Introduction to the system requirements. The chapter describes the purpose, scope and overview of the Software Requirements Specification.

Chapter 4 Overall Description

Describes the background for the requirements, a short introduction to the system, what functions needs to be supported and constraints put on the system.

Chapter 5 Specific Requirements

Specific system requirements that gives a precise meaning of what needs to be implemented.

Part II Architecture and Design

Chapter 6 Introduction to the Architecture

Introduction to the system architecture and the various services it will consist of.

Chapter 7 Service Descriptions

Describes the architectural approaches for the services in the system.

Chapter 8 Requirements Editor Component

Sketches a proposal for a Requirements Editor.
Part III Implementation

Chapter 9 Foundation Components
Describes the foundation components that were used in the project.

Chapter 10 Shared Internal Services
Describes the implementation of the Shared Internal Services.

Chapter 11 Domain Services
Describes the implementation of the Domain Services.

Chapter 12 Application Services
Describes the implementation of the Application Services.

Chapter 13 Client Services
Describes the implementation of the Client Services.

Part IV Evaluation

Chapter 14 Problems Encountered
Describes the various problems we encountered during the project.

Chapter 15 Related Work
Projects that in one way or another is related to our work.

Chapter 16 Our Contribution
Our contribution to the research community.

Chapter 17 Future Work
Proposals to future work on this project.

Appendix

Appendix A Glossary
The glossary explains some of the concepts that are used in the project.

Appendix B Maven2 Command Reference
Provides a command reference for using Maven2 with our project.
Chapter 2

Method

2.1 Project Management Process

We choose Scrum [43] as the management process for this project. Scrum is a light-weight process for managing and controlling development projects. Scrum appeals to us since it provides a loose management framework for teamwork that is easily integrated with processes and methodologies for software development, such as eXtreme Programming [54] or even Rational Unified Process for software development. Scrum is very suitable for small projects, however, it may also be adapted into larger projects.

The basic Scrum process consists of the following parts, as shown on Figure 2.1:

1. **Product Backlog**, this is the complete set of features or changes that are to be implemented in the development project. This product is created in the *Pregame* phase [43].

2. **Sprint Backlog**, this is the set of sprints (development iterations) that are to be performed during the project. Each sprint is mapped to one or more features from the product backlog that are to be implemented during the sprint. This product is created in the *Pregame* phase [43].

3. **Sprint Loop**, each sprint lasts a specific period (typically between two and four weeks). The sprint main loop is complemented by a daily Scrum meeting, in these meetings the project participants meet to discuss sprint progress and eventual problems. It is important that these daily meetings are as short and concise as possible, not a long formal meeting! The sprint is performed in the *Game* phase [43].

4. **Product Increment**, at the end of a sprint the project should have produced a shippable or deployable product increment containing the features selected for the particular sprint. After this completion a new sprint can commence. The product increment is completed and deployed in the *Postgame* phase [43].
2.2 Software Development Process

Since the Scrum process is very general, we choose to complement Scrum with a more specific software development process, namely eXtreme Programming (XP) [54]. XP will be utilized within the sprint phase of the Scrum process. XP is a light-weight and agile software development. XP builds on well-known industry best practices that are taken “to the extreme”, these practices are listed below:

1. *Simple Design*, XP puts emphasis on the implementation, the design should just outline the core features of the product. The final design will be a product of implementation and refactoring along the way.

2. *Continuous Testing*, to enable rapid and secure implementation all software units should have a prewritten test. The test will ensure that the unit performs well, even after refactoring (the code is refactored, but not the test!).

3. *Continuous Refactoring*, since the design is very rough the software units will most likely need to be refactored several times along the implementation process. This is important to enable the creation of the most beneficial solution.

4. *Pair Programming*, the developers should implement the program in teams of two, where one developer creates the code and the other watches and verifies the code.

5. *Collective Ownership*, the products of the project are owned by the entire project group. In this way everyone can modify any code if necessary.

6. *Continuous Integration*, the finished units should be continuously integrated into the final product. This enables continuous integration testing, and may also enable the product to be delivered to the customer part by part.

7. *Coding Standards*, to create clean and understandable code it is vital that the implementation team agree upon a common coding standard for the project.
8. **Small Iterations**, each iteration in an XP process should be as small as possible. Small releases are easier to test and enables a higher success rate for implementing the release on time.

We have tried to follow the XP process as closely as possible, however, we had to make a few adjustments along the way. XP is developed for use in business situations, so everything does not fit in well with projects focused on research. For example, XP requires the presence of a customer or end user in the development team. Our project does not have any customers so to speak, so this was skipped. Furthermore pair programming was only used for implementing the core framework along with the more “exotic” features, requiring special attention. We also set more focus on the design phase, since the architecture and design of the framework has great priority.

### 2.3 Tools

In the prototype that will be implemented in this project there is much emphasis for cross-platform compatibility. This is something we will try to adhere to as much as possible also with respect to the tools used to document and implement the system. Also we will be developing in a multi platform environment, so good platform independent tools are required.

#### 2.3.1 Integrated Development Environment (IDE)

The Open Source and freely available Eclipse Project \(^1\) provides an excellent IDE for Java development. This tool will be the core tool used for working with the CVS version control system, documentation and coding in the language of choosing.

Eclipse is also extremely extensible; by installing plug-ins Eclipse can be customized to support any number of different programming languages, development frameworks and supporting tools. Another advantage with Eclipse is that it is written fully in Java and therefore is available for many different software / hardware platforms.

A commercial alternative to Eclipse is IntelliJ IDEA from JetBrains\(^2\), IDEA provides an excellent and powerful IDE for Java development.

We will also make use of the Maven\(^3\) project organization / build tool from the Apache Foundation. Maven2 is an important tool for this project and the organization of the source code produces, it is described further in Section 9.2.

\(^1\)http://www.eclipse.org/
\(^2\)http://www.jetbrains.com/idea/
\(^3\)http://maven.apache.org/
2.3.2 Version Control

The freely available Concurrent Versioning System (CVS)\(^4\) provides support for centralized and secure storage of project data files. CVS enabled multiple developers to access the project files from any location, while file changes are managed and logged automatically by the system. All source artefacts created in the project will be made available in the CVS system.

2.3.3 Modeling

For modeling using the UML\(^5\) notation, the free community version of JUDE\(^6\) does a good job. JUDE is a simple but powerful cross platform modeling tool written in Java for Java development.

For more ad-hoc modeling OmniGraffle\(^7\), a commercial all-round modeling tool for Mac OSX, provides excellent facilities for creating nice looking models. The BPMN\(^8\) stencils we have used for this project can be downloaded at OmniGroup’s extras webpage\(^9\).

\(^4\)http://www.nongnu.org/cvs/
\(^5\)Unified Modeling Language
\(^6\)http://www.esm.jp/jude-web/index.html
\(^7\)http://www.omnigroup.com/applications/omnigraffle/
\(^8\)Business Process Modeling Notation
\(^9\)http://www.omnigroup.com/applications/omnigraffle/extras/content.html
Part I

Requirements
Chapter 3
Introduction

This *Software Requirements Specification* (SRS) is loosely based on the IEEE 830-1998 [44] standard, describing recommended practises for Software Requirements Specification. We have chosen to include brief Use Cases for client applications of the system in Section 4.2, to better support communication among stakeholders and give a quick introduction to the different parts of the system. This SRS for the system being specified is considered to be a part of the project, though it is not a common artefact of the XP software development process which is described in the Methodology chapter.

3.1 Purpose

The purpose of this SRS is to give the stakeholders and the developers an unambiguous and complete understanding to what are the requirements of the system being developed. It should help the developers understand exactly what the end-user wants and provide the benefits of reducing the development effort, estimate costs and schedules, provide a baseline for validation and verification and serve as a basis for later enhancements and maintenance.

This specification is written by Jørgen Rygh and Even André Fiskvik, as a part of the master thesis in subject TDT4900. The stakeholders for this project are developers of software systems with interest in the Semantic Web, researchers in the Semantic Web area of research and Workflow Management System (WFMS) A users utilizing Web Services.

3.2 Scope

The software described by this SRS is a framework and a proof-of-concept system that uses the framework. The primary goal of this proof-of-concept system is to provide a way to integrate WS-BPEL with Semantic Web Services. A secondary goal is to make relevant parts of the system generic and independent, so other systems that not are semantic aware can make use of it.
The users of the system consists of end-users using a Workflow system that uses regular Web Services and developers of software that wants an easy migrating path for their existing systems to the Semantic Web Technology. The end-users will typically interact with the system through client applications, whose implementation is not covered in this project.

The software will rely on communication with external components by using well established and emerging standards. The system can be seen on as a middleware service, providing a bridge between conventional WfMSs and the Semantic Web. The WfMSs and other clients to the system can be used to perform the tasks outlined in the scenarios from the prestudy project [16], see Chapter 4 for a detailed review of the scenarios.

3.3 Definitions, Acronyms, and Abbreviations

For a list of definitions, acronyms and abbreviations used within this document, the user is directed to the Glossary in Appendix A.

3.4 Overview

This SRS is divided into three chapters. The first chapter gives a preface to the purpose and the scope of the system – the second outlines the system in an overall description and sets the framework condition for the specific requirements finally elicited in the third and last chapter of the SRS.
Chapter 4

Overall Description

This section describes general factors that affect the product and its requirements. It provides only a background for the requirements, which will be defined in detail in the Specific requirements, Chapter 5. Section 4.1 describes the constraints put on the system with respect to the different external systems and the environment the system will need to operate in. The main functionality the system will need to address is well described in Section 4.2 with the use of textual Use Cases. The remaining section of this chapter, Section 4.3, describes the users of the system and their characteristics.

4.1 Product Perspective

One of the goals of the prototype, which will be developed during this project, is to provide a link between the Semantic Web and traditional client applications. The client’s to this system will typically be information and service oriented applications used by knowledge workers [13]. I.e. people that work with computers for information processing and information discovery tasks. The system will be a Semantic Web Service Gateway (SWSG) that provides access to Semantic Web Services. The knowledge workers of today often rely heavily on the services available on the Web to do their daily work, however, using the Web Services of the present requires much user interaction. The user must manually find the service or services that can perform his or her task and then manually execute them and handle the results of each service. Semantic Web Services enable automatic discovery and invocation of services based on semantics. This allows the creation of intelligent clients, or agents, that are able to find the best services for the task at hand based on the user’s predefined preferences. I.e. the use of Semantic Web Services enables more intelligent clients or agents, that can improve the efficiency of the knowledge workers.

Briefly described, the system will combine technologies from the Semantic Web with existing Workflow technologies, to support various tasks performed by Workflow system users utilizing Web Services. This will be achieved by letting the Workflow Management System (WfMS) interact with the framework by using built-in functionality for calling Web Services described by WSDL.
To accomplish the secondary goal mentioned in the Scope section, the system will consist of services that will be usable both as a whole and as standalone services, by making the services available through standards based Web Service technologies, such as WSDL.

The prototype will depend largely on third party components, and will function as a bridge between Workflow systems and the Semantic Web. Since the standards for the Semantic Web is still evolving, the system should try to be as technology independent as it can get. WS-BPEL will be used as the Workflow engine in the implementation of the proof-of-concept system, but it should be possible to change to other Workflow systems without much hassle, as this only depends on the implementation of the client. The system itself should be able to expose its services to any client or WfMS. WS-BPEL was chosen for the prototype implementation since it seems to be the most promising Workflow technology at the time being.

Graphical user interaction will be provided through the internal GUI\(^1\) of the Workflow system, but there will also need to be an additional GUI for defining the various requirements of the Semantic Web Service task to be performed. The interface to the Workflow systems are provided through standards based Web Services. An overview over the systems interaction with other systems can be seen in Figure 4.1. This figure was created during research in our prestudy [16], and shows the relation of the actors to the system. As we can see from the figure, the user will create the workflow (specify tasks) as usual through the WfMS.

In addition, the requirements for the Semantic Web Service tasks will be defined by the user in the “Requirements Editor” component, explained in the prestudy [16]. The “Requirements Editor” will then inject the proper Web Service calls to the Semantic Web Service Gateway (SWSG) component by modifying the workflow created by the user. The Workflow Management System will then execute the workflow, which again calls the SWSG component by using regular Web Service calls injected by the “Requirements Editor”. The SWSG component will thus function as a proxy to the Semantic Web. This requirements specification document will cover the “Requirements Editor” component and the SWSG component shown in the diagram. Only the requirements and the design, and not the implementation, of the “Requirements Editor” will be covered in this project due to time constraints.

\(^1\)Graphical User Interface
4.1.1 **Workflow Management System interface**

The interfaces for interacting with the external systems is by Semantic Web standards and the WSDL 1.1 specification [8].

4.1.2 **Semantic Web Interface**

There are no universally established specifications for Semantic Web Services, thus the interface to the Semantic Web must be established so that it can easily be changed.

4.1.3 **Software**

The system needs to operate with the following software constraints:

**Java Runtime Environment**

- Name: Java Runtime Environment
- Version number: 5.0 or greater

The Java platform is specified as a constraint to the system in order to meet the requirements of supporting multiple platforms.
4.2 Product Functions

This section presents a selection of Use Cases that shows tasks that clients of the SWSG prototype system implemented in this project can be used to achieve. The scenarios are the same that were described in the prestudy project [16]. In general the scenarios consist of tasks that are commonly solved using the Web and Web Services. With the current Web technologies such tasks often require a lot of user interaction, especially when performing more complex tasks. The SWSG system will enable the creation of more intelligent clients that can use machine reasoning to automatically discover, select and invoke services. With this increased level of automation the user should only need to define a single request, that is passed on to the back-end SWSG system. This request will describe the tasks that the user wishes to accomplish using Ontology concepts. Since working with ontologies often is a tedious task, the creation of these requests should be supported by a graphical tool (a Requirements Editor).

To describe the scenarios we have used plain text along with a listing of the main concepts related to the scenario. The concepts described in the scenarios are visualized along with their relationships as a simple UML Domain Model [27] on Figure 4.2.

![UML Domain Model](image)

Figure 4.2: UML Domain Model showing the concepts described in the scenarios.
4.2.1 UML Domain Model Description

**Requesting Actor**  Requests a result that is provided by one or more activities. Note that a requesting actor also may be a providing actor for the scenario.

**Providing Actor**  Provides services that may be used by the activities to produce a result.

**Activity**  Defines a task in a workflow that creates a result or a part of a result for the requesting actor. An activity should be seen on as an abstract service that will be populated by suitable services when these are found.

**Requirement**  Quantitative properties that the services used by the activities must fulfil.

**Criteria**  Qualitative properties that are used to rank and select the services required by an activity, or to rank the results provided by a service.

**Service**  A service is an individual service instance that provides the results that are used by the activities to provide the final result to the user. These are not specified in our scenarios, as their functionality is described by the activities.

**Result**  This represents the final result(s) given to the requesting actor.

4.2.2 S01: The Travel Scenario

The travel scenario is a common scenario with regards to Semantic Web Services and the use of automatic agents, it has been discussed earlier in multiple articles such as McIlrath, Son and Zeng 2001 [30] and Cecilie Åberg’s thesis on sButler [4].

Travels are becoming more and more common, although not specific for the knowledge worker, the travel domain provides an interesting area for the application of Semantic Web Services.

<table>
<thead>
<tr>
<th>Requesting Actors</th>
<th>Traveller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Providing Actors</td>
<td>Transport Agency, Accommodation Agency, Travelling Agency (package deals)</td>
</tr>
<tr>
<td>Activities</td>
<td>Request transportation, Request accommodation</td>
</tr>
<tr>
<td>Requirements</td>
<td>“Abstract” travel plan with departure and destination locations and dates</td>
</tr>
<tr>
<td>Criteria</td>
<td>Typical evaluation criteria are cost, flexibility and other traveller preferences and requirements (could be almost anything)</td>
</tr>
<tr>
<td>Result</td>
<td>One or more concrete travel arrangements, for user approval (or the system could automatically select the best plan)</td>
</tr>
</tbody>
</table>

Table 4.1: Elements in the Travel Domain.
The traveller often has specific preferences regarding transport, accommodation and last but not least price. The process of finding the best travel and accommodation is often a tedious process. A travel typically includes several possible means of transport and accommodation, and the traveller must manually contact multiple agencies in order to arrange the best possible travel and accommodation arrangement.

Using Semantic Web Services that utilize automatic discovery of services related to the tasks required, and that automatically invoke and evaluate the results of these services, the user can find the best available travel simply by declaring his or hers goals and preferences.

The travel domain consists of the elements specified in Table 4.1.

4.2.3 S02: The Supply Chain

The Supply Chain is also a common scenario in the Semantic Web Service domain. And one that is very relevant for experimenting with business to business (B2B) interoperability.

The supply chains of modern day businesses are often complex, including parts from multiple suppliers all over the world. Many manufacturing businesses also have a “no warehouse” policy, meaning that they produce just as much as the market demands, instead of producing for a warehouse. In order to follow the often rapid movements of the market, the business must have a flexible and adaptive production process, and thus also a flexible and adaptable supply chain.

A typical production process depends on deliveries from multiple suppliers. The business often has several preferences and demands regarding the quantity, quality and specifications of the parts or goods that are to be bought. Price is also vital, since the price of the organization’s products is closely related to the price of the parts used to create the product. The suppliers may offer better prices for certain quantities, this must be calculated into the overall process in order to get the optimal price without buying more than necessary. Other important aspects are; the location of the supplier, how fast can the parts be delivered, if the supplier can provide the needed quantity, etc.

<table>
<thead>
<tr>
<th>Requesting Actors</th>
<th>Company departments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Providing Actors</td>
<td>Company departments, Transport company, Suppliers, Customers, Government Agencies (e.g. customs)</td>
</tr>
<tr>
<td>Activities</td>
<td>Check product demand, Find product supply, Request transportation, Request parts, Check customs</td>
</tr>
<tr>
<td>Requirements</td>
<td>“Abstract” supply chain workflow specifying the needed components based on customer demand</td>
</tr>
<tr>
<td>Criteria</td>
<td>Typical criteria are cost, delivery time, quantity, etc.</td>
</tr>
<tr>
<td>Result</td>
<td>Planned supply chain</td>
</tr>
</tbody>
</table>

Table 4.2: Elements in the Supply Chain Domain.
Semantic Web Services can help automate parts of this process, by using automated B2B communication, and searching for suitable suppliers using constraints and ontology based definitions. Only parts of the process should be automated, since there should always be a control of the suppliers, checking company policy and perhaps previous experiences. A goal of the semi-automated supply chain process is to enable real time adaption to market trends.

The most common elements in the supply chain scenario are shown in Table 4.2.

### 4.2.4 S03: Online Shopping

The online shopping scenario is a B2C scenario within the e-Commerce domain, one of the domains that might see a revolution in how trading will be done with semantic web services compared to how the process works today.

The process of purchasing goods on the Internet is a rather complex one. Depending on what criteria you want set for your purchase, there are different scenarios and methods of usage. There is typically not one single method that can capture all of the user’s criteria. The base process of shopping is somewhat the same though:

1. Find online shops that provides the products the user is looking for, which typically spawns a subset of new requirements:
   - (a) The store is delivering to the user’s location.
   - (b) The store has a payment method the user finds acceptable.
   - (c) The store has a delivery method the user finds acceptable.

2. Decide upon the specific model and brand the user wants to consider.

3. Compare the products found based on descriptions, user reviews, price and other criteria the user might be focusing on.

4. Add the products to a shopping cart.

5. Checkout to confirm your order and do payment, which often leads to a new manual process of either logging in as a user or register as a new user on the online shop.

The first step is somewhat a cumbersome process. You have to know the URLs of the webshops you want to consider, or use an online search engine to find webshops available for consideration. What online shops you want to consider is dependant on what you are looking for to purchase. If you are looking for several kinds of products you typically will have to go through the same process for each product that is in a product group.

There are websites today that can provide some of the functionality, but often includes only affiliate online shops, and thus have a limited subset of available shops on the Internet. The criteria that can be used for searching for products in these kinds of affiliate websites are also limited - whereas price is typically the only supported criteria for searching the products.
Semantic Web Services can help automate most parts of this process, by using both B2C and B2B communication, and searching for suitable retailers using constraints and ontology based definitions. The user should be able to confirm the kind of purchases going to take place.

The most common elements in this scenario are shown in Table 4.3.

<table>
<thead>
<tr>
<th>Requesting Actors</th>
<th>Companies, Customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Providing Actors</td>
<td>Retail Companies, Vendors</td>
</tr>
<tr>
<td>Activities</td>
<td>Find goods, Order goods, Compare products, Find related and compatible parts, Purchase goods</td>
</tr>
<tr>
<td>Requirements</td>
<td>“Abstract” supply chain workflow specifying the needed components based on customer demand</td>
</tr>
<tr>
<td>Criteria</td>
<td>Typical criteria are cost, delivery time, delivery price, quantity, brand of choice etc.</td>
</tr>
<tr>
<td>Result</td>
<td>One or more items bought</td>
</tr>
</tbody>
</table>

Table 4.3: Elements in the Online Shopping Domain.

4.2.5 S04: Automatic Scheduling

The meeting scheduling process often involves a tedious negotiation process for finding a time and place that suits all or at least as many as possible participants. The process involves communicating with both location providers and meeting participants.

This task could be automated by utilizing Semantic Web Services that access the room booking systems and the electronic calendars of the selected participants. The participants must be ranked by importance for the meeting and their ability to participate through remote facilities such as video or audio conferencing. The system can then find one or more times that suits as many participants as possible automatically.

The most important elements in this scenario are presented in Table 4.4.

<table>
<thead>
<tr>
<th>Requesting Actors</th>
<th>Meeting coordinator(s), Meeting participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Providing Actors</td>
<td>Meeting participants, Room provider</td>
</tr>
<tr>
<td>Activities</td>
<td>Find available room, Check participant calendar, Check alternative meeting arrangements</td>
</tr>
<tr>
<td>Requirements</td>
<td>Meeting duration, Participant list</td>
</tr>
<tr>
<td>Criteria</td>
<td>Typical criterias are participant preferences, participant priority</td>
</tr>
<tr>
<td>Result</td>
<td>Meeting arrangement with date, time and location</td>
</tr>
</tbody>
</table>

Table 4.4: Elements in the Scheduling Domain.
4.2.6  S05: Information Searching / Processing

The main activity of a knowledge worker is searching and/or processing information. This activity is very important for typical knowledge workers such as lawyers, students and researchers.

Searching for information on the Internet is often a complex task, since the sheer amount of data available on the Internet is enormous and is growing every hour of the day. There are multiple services available that gathers and search information but none cover every possible source.

Semantic Web Services can help provide a framework for searching multiple information sources by use of ontology based mediation between heterogeneous data formats and semantics. The HERA project [47] uses ontologies and mediation technologies extensively to solve the problem of integrating heterogeneous information systems in a single web information system. Workflow technologies can come in handy for specifying the information flow in a processing chain. E.g. information may flow from a calculating system to a reporting system etc.

The most important elements in this domain are presented in Table 4.5.

<table>
<thead>
<tr>
<th>Requesting Actors</th>
<th>Providing Actors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information seeker</td>
<td>Digital libraries, web crawlers, research journals</td>
</tr>
<tr>
<td>Activities</td>
<td>Find information on topic, Investigate information source, Process information</td>
</tr>
<tr>
<td>Requirements</td>
<td>List of topics, Processing specification</td>
</tr>
<tr>
<td>Criteria</td>
<td>Typical criteria are information age, source, etc.</td>
</tr>
<tr>
<td>Result</td>
<td>Processed information</td>
</tr>
</tbody>
</table>

Table 4.5: Elements in the Information Searching Domain.

The system will have three main functions. The functionality is identified as follows:

1. **Requirements Editor**, will be used to specify the correct mappings of inputs and outputs desired of the Semantic Web Service to be called, as well as the constraints and other requirements of the requested operation by the user.

2. **Service Discovery**, the system will need to provide an interface to the external systems for finding the desired Semantic Web Services. The services found should adhere to the user’s specifications and constraints.

3. **Service Composition**, there will often be situations where several Semantic Web Services will need to be executed in a pipe to achieve the correct result, thus the system will need to order the services in a specific order. This functionality does not need external interfaces, but is rather a functionality which should be performed before returning the discovered services.
4. Service Execution, the system will need to provide an interface to the external system for executing the discovered services.

4.3 User Characteristics

The SWSG system that will be implemented in this project will typically consist of the following two user groups:

**Developers**

Developers of clients that perform tasks that require dynamic allocation of services and information. The clients created by these developers will be typical end-user applications used by knowledge workers or other users that intensively use the Web and Web Services for developing or processing knowledge and information.

**Workflow Power Users**

Power users that are familiar with the use of WS-BPEL of some equivalent workflow technology. These users will use the system to find the best services for the workflows that perform their business processes. To accomplish this these users will need some form of requirements editor to generate the proper queries to send to the SWSG system.
Chapter 5

Specific Requirements

This chapter describes the specific requirements of the system prototype.

5.1 System Features

This section describes the specific requirements to the various features of the prototype system.

5.1.1 Requirements Editor

Introduction/Purpose of feature

The WS-BPEL specification does not support tasks that require native user interaction, so we need to make an external component that is responsible of getting input from the user. The inputs are the requirements of the action that is to be performed by invoking a Semantic Web Service, hence we have called the component that retrieves these inputs from the user a Requirements Editor. This component will not be a substitution for the user’s workflow designer/editor, but rather a complementary application. The component reads the workflow files already created by the user, and injects the correct Web Service calls to the SWSG wherever abstract tasks are found. The RE can also be a component embedded into a non workflow end-user client.

Associated Functional Requirements

1. Provide interface for reading workflow file. The component must be able to read an existing workflow file. The interface must be designed so it will be easy to extend the component with various workflow formats.

2. Provide interface for writing modified workflow file. The component must be able to write a semantic query to the abstract service invocation to the workflow file. The interface must be designed so that it is easy to extend the component with various workflow formats. The abstract tasks in the workflow file will be replaced with concrete Web Service-tasks set to call the SWSG with the proper parameters gathered from the user input.
3. **Allow user to look-up Ontology concepts.** One key feature of this component will be to provide an interface to the user that makes it easy to reference Ontology concepts used for input, output, requirements and results/effects.

4. **Allow user to define the required inputs.** The user must be able to define one or several input concepts that will be used to tell the Matchmaker what services to find.

5. **Allow user to define the required outputs.** The user must be able to define one or several output concepts that will be used to tell the Matchmaker which services to find.

6. **Allow user to define qualitative requirements.** The user must be able to add multiple qualitative requirements.

7. **Allow user to define requested result/effects.** The user must be able to define the wanted results or effects caused by the service by referencing ontology process models.

8. **Allow user to map output(s) from previous task to value for concepts.** Inputs and outputs are essential when working with workflows. For those who works with workflows, you will know that the results of one action/ are seamlessly passed on to the next action. The Requirements Editor must be able to assign the results from the previous action to a value for any given ontology concept.

9. **Set execution to automatic or semi-automatic mode** The Service Execution feature has support for three different runtime settings that will be made available to the WfMS user from a Requirements Editor. The three different modes, that are described in Section 5.1.4, are:

   (a) Execute services automatically
   (b) Execute services semi-automatically
   (c) Execute services statically

   When using a static non-automatic approach, a Requirements Editor will need to communicate with the system, and receive the desired Semantic Web Services composition. The interface to do this will be the same as the interface provided to the WfMS, through using Web Services described by WSDL. The Requirements Editor will then use the result to generate a workflow that calls the same services in the same order each time. When using a automatic or semi-automatic execution mode, only the Web Service task needs to be inserted in the workflow, thus enabling dynamic discovery and execution each time.
5.1.2 Service Discovery

Introduction/Purpose of Feature

The service discovery feature requires the system to be able to search multiple and heterogeneous service registries for Web Services providing the requested functionality. To enable this, the system must be able to specify mappings between the ontologies used to describe the users’ requirements and the ontologies used to describe the services. The system must also provide the functionality needed to evaluating service selection criteria, typically based on qualitative properties (such as performance, security, trust etc.) The discoverer uses ontology technology and automated reasoning in order to find the proper service for the job by evaluating the goals in a user request. To find services, the system needs to take a query in XML format.

Associated Functional Requirements

1. Discover services based on a query. A query interface must be designed to support generic matchmaking of services based on inputs, outputs, effects and qualitative requirements.

5.1.3 Service Composition

Introduction/Purpose of feature

The system should be able to analyze and decompose user requests into subtasks when necessary. This enables multiple services to work together thus achieving the final objective of the user. When the service composition is complete, the system will return this composition back to the calling WfMS. This feature will only be needed in cases where the user request must be decomposed by the Service Discovery feature. It is important that the composition feature is able to discover the required execution sequence for the services involved, the component should also be able to optimize the composition flow whenever possible. Based on the calling client’s preferences, the system must be able to return the results of the composition to the WfMS, so the WfMS can call the services through the system’s Service Execution feature.

Associated Functional Requirements

1. Compose flow of services that can be executed automatically

2. Rank services according to the query defined by the user.
5.1.4 Service Execution

Introduction/Purpose of feature

The architecture of the system must be flexible to allow the invocation of Semantic Web Services using all of the most common protocols and formats.

Associated Functional Requirements

1. *Execute services automatically.* The component will automatically select the best ranked service out of a set of services, and execute that service. The system may end up using totally different services from time to time, depending on the Service Composition component’s ranking of the discovered services.

2. *Execute services semi-automatically* The component will allow the user to select a service among the composed matches. How this selection might work will depend on the WfMS system in use.

3. *Execute services statically* In this case, the component will make the system act as a static bridge to Semantic Web Services.

5.2 Non-functional Requirements

The system must adhere to three non-functional requirements, presented as our main principles in our prestudy report [16]:

1. *Standards Compliance,* the system must use established and open industry standards, to ensure that the system may be deployed and used in the IT environments of today. The technology might be extended using defined extension points if the current standards lacks vital functionality required for the system [16].

2. *Platform Independence,* the system should be independent of the server infrastructure and the WfMS used. The system should be able to run on any relevant\(^1\) platform [16].

3. *Business Feasibility,* the system should be usable in a business context by enable agile business processes that can be changed in “real-time” in response to market changes [16].

\(^1\)In practice this means platforms based on *NIX along with the Windows operating systems supporting the Java programming language
Part II
Architecture and Design
Chapter 6

Introduction to the Architecture

This chapter introduces the architectural drivers and rationale behind our architecture, along with a short description of the main components in our architectural framework that is to be used for the Semantic Web Service Gateway prototype.

6.1 Architectural Drivers

The architectural drivers for the system are derived from Section 5.2, and are listed below:

1. **Standards Compliance**, the system should be compliant with established web service and workflow standards [16].

2. **Platform Independence**, the system should be able to run on any relevant\(^1\) platform [16].

3. **Business Feasibility**, the system should be usable for business use cases not just for research purposes [16].

Of these three principles the first two requires the most attention during the architectural design process. Although a good architecture based on the first two principles will considerably increase the chances of achieving compliance with the third principle.

6.1.1 Achieving Standards Compliance

In order to enable the highest possible level of compliance with existing standards and tools for non-semantic Web Services, workflow systems and so forth. The choice of which technologies and standards to support is not an architectural decision, however, the architecture selected must be able to adapt to changing technologies and standards. This is especially important in the Semantic Web research field where the standards are not yet completely established.

\(^1\)In practice this means platforms based on *NIX along with the Windows operating systems supporting the Java programming language
For the architecture to be able to support these changing technologies we will make heavy use of the *Information Hiding* [41] principle using Object-Oriented based encapsulation. This means that we encapsulate the underlying logic of the changeable components, thus providing a consistent interface to the rest of the system.

In order to establish the best possible interfaces, we have performed extensive research on existing standards in our prestudy [16], we have also studied the functional requirements of the system to make sure our interfaces expose the necessary amount of functionality.

### 6.1.2 Achieving Platform Independence

To achieve platform independence, we have chosen to use the Java programming language. As for architectural tactics supporting portability, we will employ encapsulation of any platform specific feature we might use. Although using the Java programming language this will probably not be necessary. The encapsulation principle was described in further detail in the previous section (Section 6.1.1).

### 6.1.3 Further Architectural Considerations

In addition to our main principles we also set up some architectural requirements in Chapter 8.4 *Technical Requirements*, in the prestudy report [16]. These are summed up below:

1. *Extendable Architecture*, due to the size and complexity of the proposed system we cannot hope to be able to implement every single feature of the system. This requires that the architecture is created with future extensions in mind.

2. *Support Reuse*, the architecture used should be able to support reuse, both internally and externally. This improves the lifetime of the system, since even if the system itself is outdated, some of its components might still be of use to other projects.

3. *High Level of Abstraction*, basing the architecture on encapsulation and a high level of abstraction will improve the chances of supporting reuse. Maintaining a high level of abstraction will also make it easier to create good durable interfaces for the implementations, this will make the code more flexible, since the underlying implementations may be replaced.

The strategies best suited for satisfying these requirements are, as described earlier in this chapter, encapsulation and *Programming by Interface* [20].

We will also try to design our architecture with a Service-Oriented Architecture [40, 7] close in mind. This requires that we maintain low coupling between the different components in our systems as well as providing interfaces that expose the features of the service in a business oriented way. I.e. the interfaces should expose the features from a top-down perspective, where the functionality is exposed from what the client needs the service to do. In a bottom-up perspective the interface is generated based on the actual implementation of the service and just lists all of the methods that the service can provide without regard to the client.
For performance and convenience issues we will not make our internal services completely technology neutral, internally we will use a Java-based workflow controller. However, it will be possible to expose the internal services as technology neutral Web Services (or something similar) if needed through configuration.
6.2 Service Layers

The SWSG system will have a layered architecture [3]. Each layer will provide a different level of services to the next. Figure 6.1 below, shows a layered view of our prototype system. The four service layers are:

1. **Client Services**, a layer on top of the system exposes services for specific clients (e.g. Web Service clients, RMI clients)

2. **Application Services**, services that the application or workflow logic of the system. Uses the underlying domain services to do the actual work.

3. **Domain Services**, contains the services that do the actual work in the system, such as matchmaking and invocation.

4. **Shared Internal Services**, services and domain objects used internally in the system. It should be noted that the services in this layer can be accessed by all other layers in the system, hence the *Shared* part of its name.

![Figure 6.1: Service Layers in the system](image)

---

*34*
6.3 Services

The services that populate the layers discussed in the previous section are:

1. **Client Access Service**, the main purpose of this service is to provide a unified and simple standard Web Service based interface that clients may connect to independent of their platform or implementation. This will actually be an extra layer on top of the Top Service Layer, in this sense it is actually a client of the system.

2. **Façade Service**, this will be the top level service layer for the system. It will act as a thin service layer that provides simple access to all public services offered by the system using the underlying domain services.

3. **Matchmaker Service**, is a domain service that matches the user’s requests to suitable services, known to the system.

4. **Selection Service**, this domain service evaluates the different candidate services found by the matchmaker component and selects the best service for the job.

5. **Invocation Service**, is a domain service that enables the system to call a services written in a variety of service description languages.

6. **Registry Service**, is an internal service that provides persistence services to the other domain services.

These are basically the same as the components that are proposed in our prestudy report [16] and is shown on Figure 6.2. There are, however, some changes to the components proposed in the prestudy. First off the Internal Workflow Controller has been renamed to the Façade Service, as this suits the its purpose better. The Security Component and Service Discovery Component from the prestudy are not mentioned in the list above, these components will not be developed in the prototype. The Security Manager will also not be implemented in the prototype. The Discovery Component would require the system to add features enabling discovering services on the Semantic Web by connecting to various search engines and repositories for Semantic Web Services. Since these are not yet in place, it will not be feasible to implement this at the current time. We will instead enable the users to search in a pre-defined, local service repository.

The Mediation and Translation Component has been removed as a stand-alone component or service. Instead we have chosen to integrate this into each component where such a mechanism is required. E.g. the Matchmaker and Invocation Services. The reason for this solution is that the translation / mediation services required are already tied to the functionality of each component using the service.

The Reasoner, Knowledge Base, Task Analyzer, Planner and Composition components have been merged into the Matchmaker and Selection Services. The reason for this is quite simply that most of these services are included in the matchmaker, which is a pre-packaged component. The Reasoner is independent of the matchmaker, but in practice the system will only communicate with the Reasoner indirectly through the Matchmaker.
Another detail that is worth mentioning is that we have gone from using the term “component” to use “service”. This is to underline that it is our intention to make the component reusable and that it should be able to expose each component as an independent service.

Figure 6.2: System sketch from the prestudy [16]
Chapter 7

Service Descriptions

7.1 Client Access Service

This chapter describes the architecture and design of the Client Access Service.

7.1.1 Description

The Client Access Service (CAS) provides the clients with a standard Web Services based interface for accessing our proposed system. The service must be able to provide clients with a clean and easy to understand API\(^1\), while at the same time enforcing the security requirements of the underlying system. This latter task will need the help of a Security Manager, that should be common to the entire system. The Security Manager will not be implemented in this prototype.

In the original proposal for this service as described in our prestudy report [16] also included a Service Access part, specializing on enabling the system to interact with external services. This functionality have been moved to the Invocation Service.

For the prototype implementation we have set the limitations as shown in Table 7.1 below:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>Security features will not be implemented in the prototype.</td>
</tr>
</tbody>
</table>

Table 7.1: Limitations of the prototype implementation of the Client Access Service

\(^1\) Application Programming Interface
7.1.2 Design

The CAS is an instance of the Remote Façade design pattern [18]. This pattern is suitable for use where you have a complex system or a set of complex systems that need to publish their services to remote clients. By employing the Remote Façade pattern on such a system you create a new interface that provides clients with a clear and course grained API. This course grained API should be designed in a way that enables efficient and concise access to the services bellow. The Remote Façade design pattern is somewhat similar to the Service Layer [18] and Session Façade [2] design patterns, however, these patterns also allows the presence of domain logic. We want a clean and “dumb” layer, that just delegates the method calls down to the Internal Workflow Controller described in Section 7.2.

It will be a goal to ensure that the Remote Façade is adapted to the SOA way of thinking by defining it’s interface from a top-down perspective. That is, defining the interface in a business context from the client’s point of view.
7.2 Façade Service

This chapter describes the architecture, design and purpose of the Façade Service.

7.2.1 Description

The Façade Service (Façade) will define a clear API that exposes all public operations from the underlying internal services. It will also control the top level workflow of the system. It would be feasible to perform security and performance monitoring in the Façade, since it knows all of the underlying services involved, and controls the communication between them. The Façade should be the only service in the system to know anything about the other services. The other services should be completely independent of each other, this is to prepare the system for SOA. A local Façade will make it easy to create Remote Façades on top of it, to further expose the system using various networking technologies.

For the prototype implementation we have set the limitations as shown in Table 7.2 below:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
<td>The Façade will be implemented as a POJO, designed to run on and interact with services running on the same JVM(^2). It should, however, be possible to replace it with a distributed controller.</td>
</tr>
<tr>
<td>Security</td>
<td>Security, as discussed in Chapter 6, will only be mocked. The goal of the prototype is “only” to provide a proof-of-concept implementation of the functionality of such a system.</td>
</tr>
<tr>
<td>Language support</td>
<td>The prototype will only support OWL-S, and the functionality for changing between the various languages available in an eventual future version will not be included.</td>
</tr>
</tbody>
</table>

Table 7.2: Limitations of the prototype implementation of the Façade Service

7.2.2 Design

The most beneficial pattern for the Façade Service will be Façade [20] or a Service Layer [18]. Both of these patterns provide a single point of entry for clients and may or may not provide a course-grained interface for easy remoting. The service layer is probably the best fit for this system. As defined in Fowler’s Patterns of Enterprise Application Architecture [18] it provides a top service layer for an underlying system. This top layer typically implements the application or workflow logic of the system. I.e. the logic that is application specific, meaning the underlying services can be made more isolated and reusable. This type of service layer is called an operation script [18]. A nice feature of this approach is that it enables the service layer to be replaced by a workflow system in the future, such a design will be very flexible.

The service layer for this system will not be built on top of a proper domain model, but rather on top of various domain services. The system’s domain model will be anemic and consist mainly of data containers, or value objects.
7.3 Matchmaker Service

This chapter describes the architecture, design and purpose of the Matchmaker Service.

7.3.1 Description

The Matchmaker service will handle the matching of available services to suit the requirements of the users’ queries. Hence, the Matchmaker must be aware of the services known to the system and be able to match these to the various ontologies and concepts provided by the users. This means that the Matchmaker Service must be able to handle all ontology and service description languages that are to be used with our system.

For the prototype implementation we have set the limitations as shown in Table 7.3 below:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Support</td>
<td>For the prototype we will only implement support for the OWL ontology language and the OWL-S Semantic Web Service description language.</td>
</tr>
<tr>
<td>Performance</td>
<td>The prototype implementation will not be optimized for speed or reliability. We are only interested in showing how this may work and be used.</td>
</tr>
<tr>
<td>Composition</td>
<td>Automated service composition in the Matchmaker will not be implemented.</td>
</tr>
<tr>
<td>Discovery</td>
<td>Distributed service discovery will not be implemented, i.e. no connection to external service registries etc.</td>
</tr>
</tbody>
</table>

Table 7.3: Limitations of the prototype implementation of the Matchmaker Service

7.3.2 Design

The Matchmaker will be exposed as a domain service for the system, its functionality will be used by the top Service Layer that implements the application logic (See Section 7.2). The Matchmaker itself will be a third party “off the shelf” component. Since we aim to enable our system to handle multiple ontology and service description languages, multiple matchmakers will be required (typically one for each language). To adapt the interfaces of the various matchmakers into a common Matchmaker interface for our system we will make use of the Adapter design pattern[20]. The Adapter pattern involves defining a proper interface for all Matchmakers, and then wrapping the various Matchmaker implementations into wrapper classes that implement that specific interface. Although we will only implement support for a single language (and Matchmaker) in the prototype, this approach will enable future developers to add support for further languages and matchmakers.

The Matchmaker Service will require some form of transformation strategy for transforming the global query format to the implementation specific query language. For this the Strategy [20] design pattern might be a suitable choice.
7.4 Selection Service

This chapter describes the architecture, design and purpose of the Selection Service.

7.4.1 Description

The Selection Service will enable the system to evaluate the criteria provided by the user to the properties of the candidate services found by the Matchmaker service. The Selection Service will, like the Matchmaker, need to handle multiple different ontology languages when performing automated service evaluation.

Manual selection of candidate services should also be supported, i.e. for the creation of static workflows\(^3\). The selection process will typically be the job of the clients, but the system must provide the infrastructure needed to make this possible. That is to be able to expose or save a session, until the client has selected the services that will be used.

The limitations set on the prototype implementation are shown in Table 7.4 below:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Support</td>
<td>We will only support the evaluation of OWL ontologies.</td>
</tr>
<tr>
<td>Complexity</td>
<td>Only the very basic functionality of this component will be implemented in the prototype. I.e. basic support for manual selection and very limited automated evaluation.</td>
</tr>
</tbody>
</table>

Table 7.4: Limitations of the prototype implementation of the Selection Service

7.4.2 Design

The Selection Service will be based on the Adapter design pattern [20]. I.e. a service interface defines the public methods available to external components, while the specific implementation is provided by implementing classes for each available web service language and technology.

As with the Matchmaker, the Selection Service will be exposed as a domain service for the system, its functionality will be used by the top Service Layer that implements the application logic (See Section 7.2).

\(^3\)Where the system is only called once to find the ultimate services for the job
7.5 Invocation Service

This chapter describes the architecture, design and purpose of the Invocation Service.

7.5.1 Description

The Invocation Service provides the functionality needed to invoke Web Services described in various languages, both Semantic and Non-semantic services should be supported. The purpose of the Invocation Service is to execute the services selected in the Selection Service (See Section 7.4). This requires the Invocation Service to be able to call the services in the correct order, with the correct parameters and then perform the necessary transformations on the returned data before returning it to the caller.

For the prototype implementation we have set the limitations as shown in Table 7.5 below:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Support</td>
<td>We will only support the execution of OWL-S based services.</td>
</tr>
<tr>
<td>Complexity</td>
<td>We will only implement the basic functionality of this service in our prototype.</td>
</tr>
<tr>
<td>Transformation</td>
<td>We will not implement support for advanced transformation of the results.</td>
</tr>
</tbody>
</table>

Table 7.5: Limitations of the prototype implementation of the Invocation Service

7.5.2 Design

The design of the Invocation Service will similar to the Selection and Matchmaker services make use of the Adapter [20] design pattern in order to provide a uniform interface for all execution methods.
7.6 Registry Service

This section describes the architecture and design of the Registry Service.

7.6.1 Description

The Registry Service will be a shared internal service for the system. It will provide the domain services in the system with the means for persisting and retrieving shared data. The data handled in the Registry Service will be of the types in the shared domain model, the domain model is discussed in detail in Chapter 10.1.

The Registry Service will be used by several of the domain services in the system, and it will be the only other service the domain services know of. This will be a simple service to implement, and thus we will not set any limitations on this service.

7.6.2 Design

The Registry Service will be exposed through a public interface defining all available methods. The actual work will be performed by underlying Data Access Objects (DAO) [10] that the implementation of the interface will use. DAO, is a well known design pattern for data access. Each persistent class in the domain model get a DAO. The DAO defines a interface for the available operations and implements this for the specific data access technology used.

To simplify the implementation of typical CRUD\(^4\) operations, we will use a rather new design pattern for Java 5.0 called the Generic DAO [21] pattern. This pattern uses generics to create a DAO “template” class that provides all common operations for all DAOs that extends it.

\(^4\)Create, Read, Update and Delete
Chapter 8

Requirements Editor Component

From the requirements of the Requirements Editor, we have gone through a design and prototyping phase for how this component can be implemented. Due to time constraints we have not been able to create a working prototype of the Requirements Editor, but we have outlined in detail how we see this component implemented and working with the rest of the system.

8.1 The Requirements Editor in Relation to the Entire System

The need for a Requirements Editor is in detail discussed in the Software Requirements Specification, see Part I. In this chapter we will explain how we see this component in relation to the other parts of the system by using workflow diagrams. In the SRS we specified three different execution modes to be supported for execution of Semantic Web Services; automatic, semi-automatic and static execution. The Requirements Editor’s graphical user interface will need support all of them, and the system’s workflow will change slightly from one or the other. The GUI will thus need to support all execution modes.

We have outlined the different workflows for the system depending on what execution mode is chosen. The flows are illustrated using Business Process Diagrams (BPD) in the BPMN [42] notation, which is based on a flowcharting technique tailored for creating graphical models of business process operations. The BPD is a network of graphical objects (i.e., work) and the flow controls that define their order of performance. Figure 8.1 shows how the Requirements Editor is related to the execution of workflows that calls Semantic Web Services. The Requirements Editor Process in the same diagram is utterly explained in Figure 8.2, which shows the workflow of this process.
Figure 8.1: Workflow Execution
Figure 8.2: User’s workflow interacting with the RE
8.2 The Graphical User Interface

We have made a non-executable prototype of the Requirement Editor’s GUI, with an emphasis on covering the basic features for the user. It should also be as straight-forward as possible to add the criteria for the query that should be generated to find the desired Semantic Web Services. We will explain the measures we have used in order to adhere to these principles in the following sections, by describing the main user focus areas.

8.2.1 The Main Window

We have chosen to make the Requirements Editor a document-based application. This allows any user to work with multiple workflows at the same time, and also workflows from different workflow languages. Since it’s a document-based application, the user will be familiar with opening and saving workflows, since this is common for how document-based applications work. The workflows can be opened as regular files (as long as the file is in a workflow language the system understands), and will be modified to call our SWSG upon saving. When saving the workflow, the Requirements Editor will check what execution mode is selected, and if a static execution mode is selected, the Requirements Editor will need to provide an interface for the user to select the proper service(s). A prototype of the main document window can be seen in Figure 8.3. As can be seen from the figure, the user is able to add one or more concepts to the inputs, outputs, desired effects/results and qualitative requirements. For the inputs and qualitative requirements the user will be able to set specific values or, as seen on the first input-concept on the figure, map the input of the task (the last task’s output) for each concept. To map the input of the task to a value for an concept, simple put \{input\} as value.
Figure 8.3: Graphical User Interface mockup for the main window.
8.2.2 Ontology Concept Selection

We have used the free Open Directory Project\(^1\) to organize the ontology concepts in a structured hierarchy. The Open Directory Project is the largest, most comprehensive human-edited directory of the Web. The Open Directory powers the core directory services for the Web’s largest and most popular search engines and portals, including but not limited to; Netscape Search, AOL Search, Google, Lycos, HotBot, DirectHit, and so on. We think this will benefit the users in finding relevant concepts through a familiar structure.

![Graphical User Interface mockup for ontology concept selection.](image)

---

\(^1\)http://www.dmoz.org/
A conceptual interface for choosing the inputs and outputs is shown in Figure 8.4. The interface also contains a search-field, which returns the concepts that match the user’s search. Additionally it will show the structure of the concept so the user knows it’s the right kind of concept he is choosing (e.g. Tombstone can mean both the town in the United States and the tombstone on someone’s grave).
Part III

Implementation
Chapter 9

Foundation Components

This chapter describes the implementation details regarding the foundation components used to support our system. The foundation components consist of the programming platform along with the application framework, various support services and source code organization system. In short the components that are vital to our implementation but that does not have anything to do with the actual domain logic of the system.

9.1 Programming Language

We choose to use the latest stable version of the Java Programming Language\(^1\) at the time of writing, this was Java version 5.0. Java was suitable for this project since most tools and third party Semantic Web components are available as Java components, therefore it was most feasible to use this programming language. Java also features excellent Object Orientation features and an enormous package of supporting tools and utilities. Java is also one of the programming languages that are best known to the authors of this report. Another advantage of Java is platform independence, both the language and most of the supporting tools are available to all common platforms. The latter was a great advantage to us, as we are constantly changing between the Linux, MacOSX and Windows platforms during development.

9.2 Project Organization

Maven \(^2\) is a project organization tool that can be used to manage almost every aspect of a Java software project. It is an Open Source project developed by the Apache Foundation, and is freely available for download from the Apache website.

Maven 2 provides a default directory hierarchy for software projects. Users may use their own directory structure, but this requires more configuration of Maven. When using the default directory structure, Maven provides a lot of “out-of-the-box” features; such as project building, automated testing using JUnit (or some other tool), packaging

---

\(^1\)http://java.sun.com/

\(^2\)http://maven.apache.org/
and much more. The testing and project building features were extensively used in our project, as it greatly simplified the management of code and third party dependencies.

Maven can also be used to generate a basic project website that contains information about the project, such as access to the CVS repositories containing the source code, managing the project participants, mailing lists, project documentation showing dependencies to other third party modules, test progress, JavaDoc and more. Maven also uses a plugin architecture that makes it easy to add new features.

Another great feature of the Maven system is the management of third party dependencies. Dependencies are added to the Maven Project Document Model file (POM) along with their version number, the dependencies are then downloaded from a central repository on demand when building the project. Maven 2 also introduced support for transitive dependencies, meaning that if you add a dependency to the POM file, the dependencies of this module are automatically fetched. This feature proved very valuable to our project, since we have a lot of third party dependencies, over 100 JAR files! Unfortunately most of the dependencies used in our project are rather unknown, and therefore not added to the central Maven repository. However, maven supports the creation of a custom repository.

Using Maven for the project also helps eventual future users / developers of this code base, since the Maven 2 structure is well documented and well known by developers. Maven also makes it easy to install and deploy the system in a new environment. Maven also provides functionality for generating project files for various common IDEs such as Eclipse and IntelliJ IDEA, no need to manually setup the projects for each IDE used by the developers.

A brief guide to using Maven with our project is provided in Appendix B.

### 9.3 Application Framework

This section describes how the Spring Application Framework was used to support the prototype implementation and wire the various services together.

#### 9.3.1 The Spring Framework

We have chosen to use the lightweight Spring Framework\(^3\) for Java, to act as the container for our project. The Spring Framework is amongst other things an Inversion of Control (IoC) container \(^19\), that simplifies the process of wiring components together by implementing the Dependency Injection pattern \(^19\). The Spring Framework also provides a set of helper classes that ease the implementation of Data Access Objects (DAO) using either plain JDBC or ORM\(^4\) tools such as Hibernate or Oracle TopLink. The Spring Framework also has several other beneficial features, such as support for Aspect Oriented

\(^3\)http://www.springframework.org/

\(^4\)Object-Relational Mapping
Programming [25] (AOP), declarative transaction management and multiple helper classes for common Java tasks. The Spring Framework is also available for the .NET platform, however, we have selected the Java programming language for this project.

AOP is a programming technique that typically complements Object Oriented Programming by enabling the separation of so called “cross-cutting” concerns from the application code. Typical cross-cutting concerns are logging, security and transacting management, these aspects concern multiple objects all around the application code, but does not have any direct relation to the business logic of the object. In AOP these aspects are separated into their own objects, separate from the rest of the business logic, these objects are then injected into the code at runtime as a response to certain events (e.g. a call to a specific method, or a thrown exception).

### 9.3.2 Use of the Spring Framework

Spring has been used mainly for its dependency injection features in our project. This has enabled us to enforce minimal coupling between the various services and modules in the system. The services in the system are all configured as Spring beans in the Spring XML configuration file (spring.xml). The properties that might be interesting to change for users of the system have been separated from spring.xml using Spring’s placeholder support. These settings are now in the sws4bpel.properties file, which is in the common Java Properties format and easy to change.

One of the principles of the Spring framework is that the framework should be as non-intrusive as possible. i.e. it should not be necessary for the Spring configured classes to know of the existence of the framework. For this system, the only modules that are tied to the Spring framework are the Controller Service and the Hibernate Data Access Objects.
The Controller loads the Spring ApplicationContext and uses the Spring implemented ServiceLocator. However, it is quite possible to remove the dependency to the Spring Framework by implementing the ServiceLocator interface manually. The DAO objects can also be implemented using plain Hibernate functionality instead of the Spring helper classes.

### 9.4 Logging

Logging is provided by Log4J. This is an Open Source logging framework developed by the Apache Foundation, and freely available for download from their website. The functionality in the Log4J framework is also available for most other popular programming languages such as PHP, .NET and C++.

The Log4J framework is easy to use and has good performance. It enables logging messages with multiple priority levels. Log4J implements the Observer design pattern and enables logging to multiple mediums (appenders), such as a console, a file, a network socket, etc. Both the priority levels and output mediums can be configured in an external configuration file. In our system, logging can be configured in the log4j.properties file. Please note that the default log level for our system is set to DEBUG which creates a lot of output.

### 9.5 Automated Testing

The system used the JUnit framework by Kent Beck for automating tests. JUnit is a simple testing framework for the Java programming language, it is also available for multiple other programming languages such as .NET, PHP, Ruby and C++. JUnit is Open Source and freely available from the JUnit website.

The JUnit tests for our system can be started using Maven as described in Section 9.2. It should be noted that we do not have complete JUnit coverage for all functionalities in the system, but only for the most vital functionality. The advantage of using automated testing is to quickly see how changes in one module affect other parts of the system, as well as being able to quickly approve changes within a module by using unit testing. Note that there is a separate sws4bpel.properties configuration file for the system JUnit tests.

---

6[http://www.junit.org/](http://www.junit.org/)
Chapter 10

Shared Internal Services

This chapter describes the implementation of the Shared Internal Services in the system. I.e. the support services that are used internally by the domain services.

10.1 Shared Domain Model

This section describes the implementation of our shared domain model. The domain model represents the core concepts in the system, we have used the *anemic* approach. I.e. a “dumb” domain model consisting only of containers for common data, and little or no domain logic. Martin Fowler [17] describes the anemic domain model as an anti-pattern in violation of good Object-Oriented practices. But we argue that in a SOA context, such a model will be beneficial in order to achieve minimal coupling between the various components or services in the system. Also, there is little logic that may be put into our domain objects. The anemic domain model approach is also commonly used in SOA applications, as it enables very loose coupling between the components.

The classes in the domain model is used to transport common data to and from the various services in our system. The services themselves are complete detached from each other, the domain model being the only common link except the internal Registry Service. The classes are based on the domain model derived for the scenarios in the prestudy, this domain model is shown on Figure 4.2 in Section 4.2.

The classes in our domain model are:

1. Service, representing a service known or found by the system.
2. ServiceQuery, representing a client service query.
3. ServiceResult, representing the result of an invoked service.
4. Session, representing a stored client session.
As shown on the UML class diagram on Figure 10.1 on the following page, the classes in the domain model are implemented as plain Java classes. All domain model classes implements the `Serializable` interface, to ensure that they may be stored to file or transported over a network. All classes also overrides the `equals()`, `hashCode()` and `toString()` methods to ease working with the classes.

One thing that is missing from the `ServiceQuery` class, is the specification of Quality of Service attributes connected to the query. This was not included in the Selection Service, so therefore we did not add it to the query either.

Figure 10.1: UML Class diagram showing the Shared Domain Model
10.2 Registry Service

This section describes the implementation of the Registry Service. The Registry Service, as described in Section 7.6, is a shared internal service that is used by the domain services in the system. A UML class diagram illustrating the implementation of the Registry Service is shown on Figure 10.2 on the following page.

The implementation of the Registry Service was pretty straight forward. For storage we used the lightweight *HSQLDB* \(^1\) database. HSQLDB is a small Java based database that is extremely easy to setup and install, i.e. it requires no install. It can be used for both in-memory and file based storage, and provides a standard SQL interface. We use a Hibernate DAO to manipulate the database, so it should be quite easy to change the database to something else. This can be done by modifying the settings in *sws4bpel.properties* file.

To further simplify the data access we used the *Hibernate* \(^2\) ORM tool. Hibernate enables the use of XML configuration or Java 5.0 annotations to map plain Java objects to relational database tables. We used the XML method for configuration, as this is less intrusive on the source code. We used the *Generic Data Access Object* \([21]\) pattern for implementing the DAOs for our project. This is a variation of the Data Access Object design pattern \([10]\), that enables common functionality such as finding an entity by ID, adding an entity etc. to be written once in an abstract class. This class uses the Generics functionality in Java 5.0 to allow the class to be customized when extended by concrete DAOs, like a template in C++.

We also used the Data Access helper classes in the Spring Framework. The Spring *HibernateTemplate* class enables very simple data access, since it eliminates the need to manually manage the sometimes complex Hibernate session management. The Spring Framework was also used to inject the actual DAO implementations into the Registry Service implementation.

---

\(^1\)http://www.hsqldb.org/

\(^2\)http://www.hibernate.org/
Figure 10.2: UML Class diagram showing the Registry Service
Chapter 11
Domain Services

This chapter describes the implementation of the Domain Services. I.e. the services that perform the domain logic in the system.

11.1 Matchmaker Service

This section describes our matchmaker implementation. That is, our matchmaker integration, since we are using a third party off-the-shelf matchmaker.

11.1.1 Matchmaker Component Evaluation

We ended up with two different candidates for matchmaker components.

**OWLS-MX**

A suitable matchmaker for matching OWL-S services is OWLS-MX [39]. OWLS-MX is developed by researchers at the German Research Centre for Artificial Intelligence\(^1\) (DFKI), and is available for download from their website [39] under the Mozilla Public License 1.1\(^2\).

OWLS-MX is a hybrid matchmaker, that is it supports semantic matchmaking as well as various syntactic approaches known from traditional information retrieval (IR) [26]. The IR techniques are based on analysis of the text content in the description of the services. OWLS-MX also comes with a large collection of test services (OWLS-TC) and service requests. These will be of great use when testing the OWLS-MX matchmaker. The only downside is that none of these services (as far as the authors can see) have any actual service grounding, thus they cannot be invoked.

---

\(^1\)http://www.dfki.de/web/

\(^2\)http://www.mozilla.org/MPL/MPL-1.1.html
OWLS-MX supports the OWL-S 1.0 and 1.1 specifications, and uses OWL-S for describing user requests as well as for describing the available services. The OWL-S support in OWLS-MX is provided by the Mindswap OWL-S API [33]. Pellet\(^3\) is used as the underlying OWL-DL reasoner in OWLS-MX, Pellet is open source and freely available under the MIT license\(^4\). As hinted in the master thesis by A. Kaul [24], it might be a problem that Pellet is so closely embedded into the matchmaker. I.e. it will not be easy to replace Pellet with another reasoner. The thesis [24] also notes the heavy memory requirements of the application. However, this will not be a problem in our proof-of-concept application since performance may be sacrificed for functionality.

**OWL-S / UDDI Matchmaker**

The OWL-S / UDDI Matchmaker \([38]\) is a matchmaker system developed at the Carnegie Mellon University\(^5\) (CMU). The matchmaker is available as open source under the GNU Lesser General Public License\(^6\).

The CMU Matchmaker uses a purely semantic approach in its matchmaking processes. The services registered with the matchmaker is also registered in a UDDI registry. The latter enables better cooperation with existing Web Service repositories using UDDI. When an OWL-S service is published to the UDDI repository, the service is processed by the matchmaker engine. That is each concept in the service is annotated with information regarding how it matches against existing concepts in the matchmaker ontologies. This greatly simplifies the query process since the services and concepts known by the matchmaker already are categorized and sorted. The annotated data is stored in a MySQL\(^7\) database server. This pre-processing feature leads to a computation and memory intensive registration process, the query times are, however, dramatically lowered. This extensive load during registration might become a problem in environments where services are often updated and / or new services are constantly added. The pre-processing process will also take more and more time as the repository grows.

The CMU Matchmaker uses the commercial Racer\(^8\) reasoner component. The reasoner is loosely connected to the matchmaker as an external component, this makes it possible to replace the reasoner if necessary. The UDDI registry used with the CMU Matchmaker, is the open source jUDDI\(^9\) server by the Apache Foundation.

During our testing of the CMU Matchmaker we encountered problems when adding new OWL-S services to the matchmaker. The provided example services worked fine, but we could not add other standard OWL-S services.

---

\(^3\)http://www.mindswap.org/2003/pellet/index.shtml

\(^4\)http://www.opensource.org/licenses/mit-license.php

\(^5\)http://www.cmu.edu/

\(^6\)http://www.gnu.org/copyleft/lesser.html

\(^7\)http://www.mysql.com/

\(^8\)http://www.racer-systems.com/

\(^9\)http://juddi.apache.org/
Another problem with this matchmaker is the dependencies. The CMU Matchmaker requires the Racer reasoner server running along with a MySQL database server. We would like to have our system run as standalone as possible, especially with regards to use of external servers.

11.1.2 Matchmaker Service Implementation Details

As described in Section 7.3, we will make use of the Adapter design pattern when integrating the matchmaker component into our system. The Adapter pattern can be implemented by creating a Java interface that the wrapper classes implement. It could also be implemented using an abstract class, in stead of an interface. This, however, is more of a Template Method [20] approach. The latter approach would be feasible if the various matchmakers shares some overall functionality that could be placed in the super class. A UML diagram of the matchmaker service is shown on Figure 11.1, and described in the following text.

We selected OWLS-MX (See Section 11.1.1 above) as our matchmaker component. When integrating the OWLS-MX matchmaker into our system we used the Adapter pattern. We defined an matchmaker interface called \texttt{MatchmakerService} and created an implementation of this interface for OWLS-MX named \texttt{OWLSMXMatchmakerServiceAdapter}. This did not work out quite as expected though. We encountered several problems using the OWLS-MX matchmakers, especially with their use of serialized files for storing the current matchmaker state and the compiled local ontology. The serialization and deserialization of the matchmaker state from the hard coded \texttt{registry.data} file always threw an error, and did not seem to work. I.e. we had to load all services into the matchmaker on every load! This created a major performance problem, because loading the OWLS-MX test collection consisting of about 500 OWL-S services, took 55 minutes.

Obviously we had to fix this, and since all information regarding serialization is hard coded into OWLS-MX, we had to mess with the OWLS-MX source code. We created a custom extension of their \texttt{SimilarityMatchmaker} class, based on the source code of their original matchmaker.

Our custom matchmaker implementation, named \texttt{OWLSMXSimilarityMatchmakerImpl}, provided a custom serialization scheme. Using this new implementation we managed to reduce the matchmaker loading time to about ten seconds. Note, however, that this the ten second loading time comes after the services have been loaded into the matchmaker the first time. Adding services for the first time still takes the same amount of time as before. We also tried to make this new matchmaker implementation more configurable, but the filenames used in the matchmaker component were also hard coded in other OWLS-MX components, making this quite unfeasible. It was quite difficult to understand the OWLS-MX implementation, since it frequently used hard coded information along with making extensive use of third party modules.

Also we added a matchmaker implementation agnostic service registry for our system.
This database registry stores only the URI of each service registered in the matchmaker, and is therefore usable for every matchmaker using URIs.

Figure 11.1: UML Class diagram showing the Matchmaker package
11.1.3 Matchmaker Data Flow

The data flow between the various objects in the matchmaker service is shown on Figure 11.2. The instance of the \texttt{MatchmakerService} interface is created and configured by the Spring Framework, and then available to clients through the \texttt{ServiceLocator} interface. The custom made OWLS-MX implementation and the instance of the \texttt{RegistryService} is also created by the Spring Framework and then injected into the \texttt{MatchmakerService} instance using dependency injection.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{uml_sequence_diagram.png}
\caption{UML Sequence Diagram showing of the Matchmaker Service}
\end{figure}

On initiation the \texttt{OWLSMXMatchmakerServiceAdapter} instance will call the \texttt{load()} method on the \texttt{OWLSMXSimilarityMatchmakerImpl} instance in order to load previously registered services into the matchmaker memory. Note that this functionality can be disabled by altering the project configuration file (\texttt{sws4bpel.properties}).

If there are any new services to add to the matchmaker, these will be added by calling the \texttt{registerService()} method, this method takes \texttt{Service} objects as a parameter. Note that the services should be added to the \texttt{RegistryService} before the service is added to the matchmaker.
To perform a service match, the client must call the `performMatch()` method on the matchmaker service and pass along an instance of `ServiceQuery`. The contents of the query is then read and transformed into the native OWLS-MX format by the service. The transformed query is stored in a temporary file, and the URI to the file is passed along to the matchmaker implementation which performs the actual matchmaking based on the services registered in the matchmaker.

The matchmaker implementation returns a `SortedSet` containing the identifiers of the matching services. The `OWLSMXMatchmakerServiceAdapter` instance then loops through this set and fetches the service URI by using the `RegistryService`. This is a bit awkward, and may be fixed by modifying the `OWLSMXSimilarityMatchmakerImpl` class to read the URI straight from our file based service registry. However, we did not have the necessary time to alter this. The result of the matchmaking operation is an `ArrayList` containing `Service` instances that is returned to back to the client.
11.2 Selection Service

The Selection Service was not fully implemented during this project. The implementation described below is a *generic* selection service that ranks and selects services based on the ranking values assigned by the matchmaker. A OWL-S specific selection service should have been implemented in order to rank the services based on their Quality of Service attributes.

The `SelectionService` interface specifies the operations of the Selection Service. The first operation is the `rankServices()` method that sorts the provided collection so that the best matches are first in the collection. The second operation is the `selectService()` method, which simply performs the first operation and then returns the first service in the collection.

The interface and generic implementation can be seen on Figure 11.3 below.

![Figure 11.3: UML Class diagram showing the classes in the Selection Service](image-url)
11.3 Invocation Service

This section describes the implementation of the Invocation Service.

11.3.1 Invocation Component Evaluation

This section describes the various components we evaluated for use in implementing the Invocation Service.

**OWL-S Virtual Machine**

*OWL-S Virtual Machine (OWLS-VM)* [37], previously known as the DAML-S Virtual Machine, is a runtime environment for executing and controlling the interaction between OWL-S Web Services. It is developed at the Softagents Lab at Carnegie Mellon University (CMU). It is freely available as open source under version 2.1 of the GNU Lesser General Public License at from their website [37].

OWLS-VM provides a functional API for executing OWL-S services and handling their inputs and outputs, and supports both the 1.0 and 1.1 versions of OWL-S. OWLS-VM is based on the CMU OWL-S API, which is not compatible with the OWL-S API implementation from Mindswap. This is a disadvantage since the OWLS-MX 11.1.1 matchmaker uses the Mindswap API.

An invocation service using OWLS-VM was implemented for the system, in the class `OWLSVMInvocationService`. However, a number of problems were encountered when testing this invocation component. We could not find any proper documentation of the project and the source code for the OWLS-VM was not available from their website. Neither the OWLS-VM or CMU OWLS API packages have been updated since 2005, thus they had several dependencies to old and outdated third party software components. This dependency to old components created a versioning conflict with other components requiring newer versions of the same components.

Another problem was that the OWLS-VM had some sort of conflict with the OWLS-MX matchmaker component. It turned out that if these two components were loaded in the same JVM\(^\text{10}\), OWLS-VM would throw an exception complaining on the naming of the service being invoked. We traced the problem to the creation of an *OWLKnowledgeBase* instance within the Mindswap OWL-S API used by OWLS-MX. OWLS-MX on the other hand, had no problems coexisting with OWLS-VM within the same JVM. The latter problem could have been fixed by separating the matchmaker and invocation services into two components that run on different JVMs (this is supported by our architecture), but we figured that it would be better to avoid this added complexity in our prototype system. Thus we discarded the OWLS-VM approach to service invocation.

\(^{10}\) Java Virtual Machine
Mindswap OWL-S API

The *Mindswap OWL-S API* [33] is a Java API that enables programmatic manipulation and execution of services described using OWL-S. It was developed by the Mindswap [32] project in cooperation with the Fujitsu Labs of America\(^{11}\) and is distributed freely under the MIT license\(^{12}\) from the Mindswap website.

Jena \(^{29}\) provides the OWL and RDF base for the Mindswap OWL-S API, and uses many of the same classes and properties as the alternative OWL-S API from CMU. Contrary to the CMU OWL-S API, the Mindswap API is well documented and also have an active mailing list available from the Mindswap website. This greatly improves the usability of the API. Since the Mindswap API is also used by the OWLS-MX matchmaker component, it enables us to have less dependencies than the OWLS-VM alternative.

11.3.2 Invocation Service Implementation Details

As described in Section 7.5 the invocation service is implemented using the Adapter \(^{20}\) design pattern. The Java interface *InvocationService*, as shown on Figure 11.4, specifies the operations available from the invocation service.

The final execution component selected for the system was the Mindswap OWL-S API [33], described in the previous section. Before using the Mindswap solution we created a test implementation using the OWL-S Virtual Machine [37]. However, as described in Section 11.3.1, we encountered several problems when using this implementation. Hence, we choose to implement the service based on the Mindswap OWL-S API.

Implementing the service using the Mindswap API went basically without any particular problems. We did, however, discover an incompatibility between the latest version of the API\(^{13}\) and the latest version of OWLS-MX\(^{14}\). One of these incompatibilities were due to a call to `OWLSKnowledgeBase.setReasoner()` where the API would throw an exception and terminate when passed a *null* value by OWLS-MX. This problem can be solved by modifying the OWLS-MX source code. But because OWLS-MX also requested some classes that were removed from the new version of the API, we did not have the neccessary time to fix all incompatibilities.

The Invocation Service is implemented in a way that completely isolates it from the other services in the system, except the use of the shared domain model classes. It can therefore quite easily be separated from the other services and be hosted on a separate machine if neccessary.

\(^{11}\)http://www.flacp.fujitsulabs.com/
\(^{12}\)http://www.opensource.org/licenses/mit-license.php
\(^{13}\)Mindswap OWL-S API version 1.1.0 beta at the time of writing
\(^{14}\)OWLS-MX version 1.1b at the time of writing
A known problem in the Invocation Service, that we did not have the time to fix during this project, is the mapping of input parameters. The ServiceQuery class contains input values mapped to generic ontology concepts. These concepts have an unique URI, and the matchmaker is able to match these concepts to related concepts by analyzing the ontology that specifies the concepts. This works fine, however, the services found by the matchmaker might not contain input parameters using the exact same URI as the initial concepts in the query, but often they are matched based on the results of logical deduction.

When fetching input parameters from the query in the Invocation Service, the current implementation checks for an exact URI match on the selected service. There are two solutions for this problem; The matchmaker can add information to the matched services regarding which input concepts that match the ones in the query. Or the invocation implementation could be altered to check what the relation between the query and service is at before adding the parameters.

Figure 11.4: UML Class diagram showing the Invocation package
11.3.3 Invocation Data Flow

The data flow between the various classes that make up the Invocation Service is shown in the UML sequence diagram on Figure 11.5. The data flow of the Invocation Service is quite simple, when the service is created by the Spring Framework a client can fetch it through the ServiceLocator interface.

![UML Sequence Diagram showing of the Invocation Service](image)

Figure 11.5: UML Sequence Diagram showing of the Invocation Service

When the client calls the `invoke()` method on the InvocationService instance the invocation service creates the necessary objects from the Mindswap OWL-S API, and parse the service description using the functionality in the OWL-S API. The input parameters from the ServiceQuery object are extracted, and the input values (consisting of a concept URI and a String value) are inserted into a Mindswap ValueMap object which is passed along when the `execute()` method is called on the Mindswap ProcessExecutionEngine instance. The resulting ValueMap containing the output data is then translated into a ServiceResult object which is returned to the client.
Chapter 12
Application Services

This chapter describes the implementation of the Application Services of the system. I.e. the services that implement and coordinate the main application and workflow logic in the system.

12.1 Façade Service

This section describes the implementation of the Façade Service, which is the top Service Layer [18] of the system.

12.1.1 Façade Service Implementation Details

The typical way to implement a Service Layer in JEE\(^1\) is to make use of a EJB\(^2\) Session Bean. We have, however, decided to make use of the more lightweight approach based on POJOs\(^3\) in the lightweight Spring JEE application framework (See Section 9.3).

As seen on Figure 12.1, the Façade Service is defined by the SWS4BPELService interface. This interface is implemented by the SWS4BPELServiceImpl class, which is a plain Java class (POJO).

The Façade Service, as introduced in Section 7.2, exposes the Application Services of the system. The interface exposes these services as quite course grained methods, in order to simplify the implementation of Client Services. In addition to this, the SWS4BPELServiceImpl class also initializes the Spring Application Context. The Spring Application Context parses the Spring configuration file, and provides access to the Spring configured resources and beans. The Façade Service uses the Application Context to get an instance of the Spring implemented ServiceLocator, which provides access to the Domain Services as well as the Registry Service.

---

\(^1\)Java Enterprise Edition
\(^2\)Enterprise JavaBean
\(^3\)Plain Old Java Objects, standard Java objects in contrast to EJBs that must implement a variety of interfaces and have a very specific packaging
12.1.2 Façade Service Data Flow

The data flow of the Façade Service represents the overall data flow of the entire system. There are two interesting data flows in the Façade Service, the first being the automated process and the second being the manual process, where the user or client performs the selection task. It should be noted that the Façade Service also provides access to the individual operations in the process.

Automatic Process

In the automatic process, as illustrated on Figure 12.2, the system performs all operations in the process in the following order: Match → Select → Invoke

The client must first get an instance of an SWS4BPELService implementation. Then this implementation will initialize the Spring Framework and fetch the domain services along with the Registry Service from the ServiceLocator. The client can now add eventual new services to the system, and the Façade Service will make sure that these new services are added to both the RegistryService and the MatchmakerService.

When all services are registered and the system is properly initialized, the client can pass the ServiceQuery of the user to the performAutomaticProcess() method. This method then calls the performMatch(), performSelection() and performInvocation() methods before returning the final result to the user as an instance of ServiceQuery.
Figure 12.2: UML Sequence diagram showing an automatic process performed with the Façade Service
Manual Process

The manual process, as shown on Figure 12.3, lets the user or client select the services that are to be invoked by the system. This is beneficial when using the system to create a static workflow, i.e. selecting the services that will be called every time.

Figure 12.3: UML Sequence diagram showing a manual process performed with the Façade Service
For the manual process the client must initialize the Façade Service in the same way as for the automatic process. However, when doing the actual work the client must call the `performManualProcess()` method, which performs a matchmaking process and stores the results of this along with the user’s query in a Session. The client then receives the identifier of this stored session, which is later used when calling the `resumeManualProcess()` method to invoke the selected service and return the result. The service selection must be handled by the client, the stored Session object can be retrieved from the Façade Service using the `performSessionLookup()` method.

It should be noted that it is also possible for the client to store the session data, and call the individual methods for matchmaking and invocation.
Chapter 13

Client Services

This chapter describes the implementation of the Client Services. I.e. the services that provide remote access for the clients.

13.1 Client Access Service

This section describes the implementation of the Client Access Service.

13.1.1 Client Access Service Component Evaluation

For the Client Access Service we only evaluated one component, namely the XFire\(^1\) Web Service framework for Java. Relevant alternatives to XFire could have been Apache Axis\(^2\), Spring Web Services\(^3\) or exposure through an ESB such as Mule\(^4\). XFire is a rather new framework, but it is quickly gaining momentum within the Java community, since it is very easy to use and it provides good integration with the widely used Spring Framework and Acegi Security Framework\(^5\). XFire is developed by Envoi Solutions LLC, but is distributed under a flexible Open Source license making it freely available for deployment, modification and redistribution.

XFire enables us to expose Java POJOs\(^6\) as WSDL Web Services by using either Java 5.0 annotations or XML configuration. The annotation approach is based on the JSR-181 specification for Web Service annotations, this will also be an official feature in the upcoming “Mustang” version of Java, under the Java extension called \texttt{javax.jws}.

The WSDL for the service is generated based on the provided annotations or XML configuration as well as general information retrieved using Java reflection. The downside of XFire is that the resulting XML input and output formats for the services tend to get messy when returning or retrieving complex types.

---

\(^1\)http://xfire.codehaus.org/
\(^2\)http://ws.apache.org/axis/
\(^3\)http://www.springframework.org/spring-ws/
\(^4\)http://mule.codehaus.org/
\(^5\)http://www.acegisecurity.org/
\(^6\)Plain Old Java Object
XFire can be deployed on any Java based Web Container, such as Apache Tomcat or Jetty.

13.1.2 Client Access Service Implementation

The Client Access Service provides remote clients with a standard WSDL based Web Service interface to the application services, i.e. the Facade Service. The XML method for exposing services is a completely non-intrusive method for exposing POJOs, however, however, we choose the annotated approach for its clarity and simplicity. For this we created a new web project with Maven consisting of a single class, SWS4BPELWebService, as shown on Figure 13.1 below. This class is configured using the Spring Framework and has the Facade Service, SWS4BPELService, injected using Spring dependency injection. The methods in the web service class is annotated using the JSR-181 annotations and reroutes the calls to the Façade Service with performs the actual work of delegating the tasks among the underlying domain services.

We created a new project for this service, since it seemed logical to distribute the core services as a Java Archive (JAR) for easy reuse. The Client Access Service is distributed as a Java Web Archive (WAR), this single WAR file contains all necessary dependencies and can easily be deployed on any compliant Java Web Container. In our tests we used Apache Tomcat\(^7\) 5.5 running on a JBoss\(^8\) 4.0 application server.

![Figure 13.1: UML Class diagram showing Client Access Service](http://tomcat.apache.org/\[\]

![Figure 13.1: UML Class diagram showing Client Access Service](http://www.jboss.com/\[\]

\(^7\)http://tomcat.apache.org/

\(^8\)http://www.jboss.com/
The data flow of the Client Access Service is not documented in this report, as it simply passes all method calls on to the top level Façade Service. See Section 12.1 for the details regarding the Façade Service implementation and data flow.
Part IV
Evaluation
Chapter 14

Problems Encountered

The main problems encountered in this project was in relation to the architecture phase. We also had a bit of struggle with the implementation – for almost every solution, several new problems surfaced. Many of the problems were related to the current state of the Semantic Web and the state of the components used. At the time of writing there is no complete or standardized environment for managing Semantic Web Services, which lead to some of the problems encountered, which are described in the following sections.

Lack of Documentation

One of the most common problems we encountered was the lack of updated technical documentation for the tools we were using. The research based tools often provide excellent research papers explaining what the tools will do along with the scientific theories on which the tools are based, but lack straightforward technical documentation explaining how to setup and use the tools.

Examples are also often missing, a positive exception is the Mindswap OWL-S API that comes with excellent example code and documentation.

We also encountered projects where no technical documentation, or implementation was available to the public. This was the case with the WOSE [55] and sButler [4] projects.

Incompatibilities

We encountered several incompatibility problems. In addition to the problems with incompatible libraries we also discovered that all tools did not work on all OWL-S services. For example the OWLS-MX matchmaker had some problems working with the test services provided by the OWL-S API.

Also the OWLS-MX test collection could not be executed, however, this was simply due to the lack of an actual WSDL grounding for those services.
Missing Dependency Information

This is actually related to the first point regarding a general lack of documentation. Most of the projects we encountered during this project made use of several third party software libraries. This is very common in the Java and Open Source communities, since reuse is one of the main principles in Object Oriented programming. However, the problem we encountered was the lack of documentation regarding the libraries that were used. Most projects did not provide any information regarding the version of the libraries used, they simply included all the dependency libraries as JAR files in the project distribution.

When several projects also share the same dependencies, knowing which versions of the libraries that can be used is critical. We encountered problems when two different libraries required different version of the same library (e.g. OWLS-MX and OWLS-VM), the result was that we were required to replace one of the libraries with another.

Using Maven 2 we managed to keep track of the dependencies used in our project, however, most of the libraries we are using are not in the common Maven 2 repository. So we had to create our own repository, this repository is located on the provided CD. Some of the dependencies in the Maven 2 repository might have weird version numbers, this simply means that we never managed to figure out which version that was used. A version number like OMX simply means that the library is the same that was bundled with the OWLS-MX distribution.

Lack of working technology

A lot of the tools we encounter are unfinished and rather rough around the edges. This is of course natural when dealing with such new and experimental technology, hopefully the tools will mature when the technologies and standards become more stable. Many of the tools we have investigated are the results of small student projects, the problem with these tools are that they are typically abandoned as the student projects come to an end. Often they are also abandoned as unfinished tools.

The other main type of projects we have encountered are the massive international research projects, such as WSMO [51] and METEOR-S [31]. These are constantly improved and modified and typically produce a vast amount of non-technical documentation. I.e. a lot of good research papers, but no simple technical documentation explaining how to use the tools they produce.

In the early phase of our project we did some initial testing of technology that could be used as a part of our solution. We tried to deploy and test the WSMO project, but could not find any solution on how this could work properly. This might albeit be because of the lack of proper technical documentation for the project as well, as described in previous section.
New Concepts and Untested Tools

The concepts and technologies for the Semantic Web are still rather new and mostly untested for real life scenarios. We especially discovered this when we began to connect the various tools. It seems like most tools were created and tested within their own environments, isolated from the rest of the world. So when you try to put different tools together in the same environment you literally open Pandora’s Box!

The Semantic Web technologies has a lot to offer, but it has been difficult to figure out how to use all these new technologies and concepts to their full potential. This is typical for all new technology, it takes a while before they mature. A lot of research along with testing the tools and technologies in business oriented projects is needed.

Both the tools used and the ways these tools are utilized will require a lot of practical testing before they can be called mature. This is something which will only come with time.

Lack of Time

As with all projects, time has been an important limiting factor. Most theoretical information was gathered during the prestudy project [16], but the practical aspects of the projects were not covered to any great detail in the prestudy. Hence, much time has been spent on experimenting with the various tools available, in order to figure out which tools that can be used and how to use them.
Chapter 15

Related Work

This chapter describes some projects that in one way or another are related to the work of this project.

15.1 Web Service Modeling Ontology

The Web Service Modeling Ontology (WSMO) [52] is a project undertaken by the WSMO Working Group at the SDK Cluster\(^1\). The purpose of WSMO is to create an ontology for the WSMF, that describes the various aspects of Semantic Web Services with the goal of simplifying application integration [52]. WSMF, or the Web Service Modeling Framework [46] is an earlier project that defined a framework for supporting the development, discovery and description of Web Services and Web Service compositions [15]. The main principles of the WSMF was Maximal Decoupling and Scalable Mediation, and these principles are also used as the base for the WSMO project.

The WSMO project also has two subprojects, the Web Service Modeling Language (WSML) [51] and the Web Service Execution Environment (WSMX) [50]. WSML is the ontology language used by the WSMO project, it provides a formal and rule based method of describing Semantic Web Services and ontologies. WSML is based on several logical formalisms, namely Description Logics, First-Order Logics and Logic Programming. This is useful for enabling automatic reasoning around WSMO concepts and provide efficient mediation. WSML exists in five variants, where each variant has a different level of expressiveness. These are WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full, the latter being the most expressive [12].

WSMX is the reference implementation of the WSMO system. It provides an implementation of a complete Semantic Web Service environment based on the WSMO ontology and the WSML language. WSMX will provide a framework capable of supporting dynamic discovery, mediation, selection, invocation and composition of Web Services [50].

\(^1\)A Cluster of three European research projects in the area of Semantic Web and Semantic Web Services; DIP, SEKT and Knowledge Web
The WSMO Working Group, that manages the WSMO projects, have submitted the WSMO, WSMX and WSML as Member Submissions to the W3C [56].

WSMO is one of the projects that were investigated in the prestudy project [16]. Early in this project we investigated the possibility of using WSMO, or rather the WSMX implementation, as the base for our system. However, we encountered several problems. Although there are many reports available on this project, most of them are research oriented, and does not provide any specific technical information. The WSMX implementation is a huge Java based system, and quite difficult to understand without any direct insight or clear technical information. We spent quite some time trying to figure out how to incorporate WSMX into our system, but at the end we decided to look for alternatives. Another problem with WSMO is the custom description language it uses, WSML, this language is only used in the WSMO project.

The WSMO project should, however, be watched in the future as they seem to be making considerable progress in solving many of the problems related to the practical use of Semantic Web Services.

15.2 Workflow Optimisation Services for e-Science

The Workflow Optimisation Services for e-Science (WOSE) [55] is an ongoing project at the University of Cardiff\(^2\). The main goal of the WOSE project is to develop an application-level service that will enable grid based applications to dynamically allocate grid services at runtime [55].

To achieve dynamic selection of grid services the WOSE project has chosen an approach similar to our dynamic workflow approach [16]. This approach uses a proxy service to act as a placeholder for the actual service, which will be dynamically discovered at runtime [23]. WOSE also uses Quality of Service (QoS) criteria to select the best service for the job. For service discovery the WOSE project provides a standard Web Service that can be called from any Workflow Management System (WFMS), this service can use three different schemes to discover services for the WFMS [23]:

1. \textit{byNAME}, uses keyword matching on the name to find a suitable service in a UDDI [35] registry [23].
2. \textit{byMETA}, uses the additional meta data in the UDDI registry to find the best service [23].
3. \textit{byONTOLOGY}, uses semantic matchmaking to find the best services [23].

To evaluate QoS criteria, the WOSE project uses an optimization service. This service can evaluate discovered services by their performance and level of trust [23].

\(^2\)http://www.cardiff.ac.uk
As for actual technology the WOSE project uses OWL-S [28] as their service description language along with ontologies written in OWL. The OWL-S services are registered in a UDDI service registry, the UDDI registry is extended with additional WOSE metadata and links to ontologies to enable the more advanced discovery schemes. WOSE uses the open source ActiveBPEL\(^3\) workflow engine and the WS-BPEL[9] workflow language as their WfMS. They also make use of an abstract workflow description language called SCUFL\(^4\).

We were in contact with the participants of the WOSE project in order to investigate how we might be able to use the results of their research in our project. The WOSE prototype is almost fully developed, but lacks the byMETA and byONTOLOGY discovery schemes. The source code for the WOSE prototype is at the time of writing this report closed source. The prototype binaries would be released for public evaluation during late March / early April 2006, this is unfortunately too late for our project to make any usage of the WOSE prototype itself.

15.3 METEOR-S

METEOR-S [31] is the follow on project to the METEOR\(^5\) project at the LSDIS\(^6\) Lab, University of Georgia\(^7\). The METEOR project focused on workflow management techniques for transactional workflows. The METEOR-S project will focus on supporting Web based business processes using SOA principles and Semantic Web technologies [31].

METEOR-S aims to enable the creation of dynamic business architectures by providing a platform for dealing with the complete lifecycles of Semantic Web Services [31, 48]. The project builds upon existing SOA, and Semantic Web Service technologies, some of which have been extended by the METEOR-S project.

The environment being developed by the METEOR-S project consists of the following five components:

1. Abstract Process Designer, that enables users to define abstract processes using an extended version of the WS-BPEL language that supports semantic annotations [31].

2. Semantic Web Service Developer / Annotator, that let users add semantic annotations to WSDL based services. The semantic annotations are added by using the WSDL-S [1] language, which is an extension to the official WSDL specification developed by METEOR-S in cooperation with the IBM Corporation [31].

3. Semantic Publication and Discovery Engine, an engine that allows annotated services to be published to a repository based on an extended version of the

\(^3\)http://www.activebpel.org/
\(^4\)http://www.gridworkflow.org/snips/gridworkflow/space/SCUFL
\(^5\)Managing End-To-End OpeRations
\(^6\)Large Scale Distributed Information Systems
\(^7\)http://lsdis.cs.uga.edu/
UDDI [36] standard. This component also enables the registered services to be discovered by external clients, by querying the UDDI repository [31].

4. **Constraint Analyzer**, enables the selection of the most suitable services for a given task. The selected services are injected into the abstract process defined by the user [31]. This component is somewhat similar to our *Selection Service*, see Section 7.4.

5. **Execution Environment**, an execution environment that enables the execution of the concrete workflow created by the Constraint Analyzer [31].

The METEOR-S project was very interesting for our project, both because of its features and because it is based on extensions to established standards. However, it is still far from being finished. The METEOR-S Web Service Annotation Framework is currently the only available component. We investigated this component early in this project, but did not find it usable without the currently unpublished components. The METEOR-S project should, however, be watched in the future.
Chapter 16

Our Contribution

This chapter discusses the various contributions of this project.

Framework for Supporting Semantic Web Service Applications

The primary goal of our system was to add support for Semantic Web Services to WfMSs. In the search for a good solution to close the gap between WfMSs and Semantic Web Services, we found the best way to do this was by using standards, and have the WfMS work with as few modifications as necessary. This lead to a system that can easily be used from any client that is able to consume regular Web Services.

Our system can also be extended with new Client Services to allow clients to connect using other protocols as well. This was an important decision for us, as it also lead to the possibility of creating a system that not only supported the primary goal, but that is able to provide a framework for supporting Semantic Web Service Applications in general. We see this as an easy migrating path for systems that currently can use Web Services, but where it might not be feasible to spend too many resources on implementing native support for Semantic Web Services. Also the system allows bringing the benefits of the Semantic Web to legacy applications, since our system can be extended to support any communication protocol.

In addition to adding Semantic Web capabilities to existing systems, the system can also be used as a back-end system for lightweight knowledge worker clients. Such clients can enable performing of the tasks sketched in Section 4.2.

Framework for Testing new Technology

One of our primary architectural drivers for the system was modifiability. Most components should be fairly easy to exchange for other components. This makes our system a good framework for testing new technology for the Semantic Web. It could for example be used for testing new matchmakers as they are made available. In our project we tested various matchmakers, two invocation components and other packages.
The framework provides simple adapter interfaces for the various components. Any new matchmaker, invoker, selection scheme, etc can therefore be injected into the system by implementing these adapter interfaces. The actual implementations of each service in the system is configured using the Spring Framework, so replacing an implementation for another is as simple as modifying the Spring XML configuration. When the new component is all setup it can be automatically tested by running JUnit tests. This provides a simple method of comparing similar tools.

Testing how Various Technologies can Interact

Our system provides a framework for testing new technologies as mentioned earlier in this chapter. Another usage for the framework is to test how various tools can interact and work together. E.g. how a specific matchmaker works with an invocation framework.

This feature is important for being able to mature the tools available. since it will help developers discover eventual incompatibilities and problems with these tools.
Chapter 17
Future Work

This chapter describes the possibilities for future projects based on the work described in this report.

Fixing Known Problems

The first priority for future work should be fixing the known problems and bugs in the current framework, these problems are listed and described in this section. The most critical issue with the current implementation is with the Invocation Service. The problem is that the concept URIs from the query does not necessarily match the concept URIs from the executable service. This problem is described at the end of Section 11.3.2. It should also be investigated how input values can be fetched from the clients, perhaps an XML Schema [11] can be passed along with the query?

Another known problem is the XML format used for the Client Access Service (See Section 13.1.2). The current format is auto generated by the XFire framework, and is far from intuitive. A more human readable format would greatly improve the ease of using the system from WS-BPEL or any other Web Service based client. To enable this, it might be necessary to replace XFire with an alternative Web Service framework.

The result of a Web Service invocation might require some transformation before being returned to the client application. The current results are returned as either RDF [14] or String data. A unified approach would be to prefer, e.g. returning all data in the RDF XML format.

Testing New Tools for the Semantic Web

The system is designed with modifiability in mind, and thus it provides room for replacing all services and components with alternates. This is a nice feature for testing new matchmakers, APIs, languages and so on for the Semantic Web. More and more tools for the Semantic Web are appearing each week, and our framework can be used to test these tools.
Improving the Framework

There is a lot of room for improvement in the system. The Selection Service (See Section 11.2 is in need of a new semantic aware implementation. The current generic implementation only provides rudimentary service ranking and selection. An implementation that performs true semantic evaluation of the services based on the Quality of Service (QoS) attributes of the services would greatly improve the potential of the system.

Security is also a missing feature of the current implementation. It should be investigated how Semantic Web applications can be secured will still remaining as flexible as possible. The security effort should especially focus on the trust and reliability issues, perhaps the emerging WS-* standards provide some features for solving these problems? For the technological security approach the Aegi Security Framework could be a suitable approach for the current system, as it is designed especially for use with the Spring Framework and Web applications.

An architectural review should be undertaken to ensure that the architectural approach used by the system are optimal. Reuse, extendibility, modifiability and testability should be the primary metrics for such a review.

In addition to adding the existing features of the system it is also possible to add new features. New domain services can be added to extend the functionality of the system. Another form of extension can be to add support for additional semantic web languages, such as WSML from WSMO or WSDL-S from METEOR-S, since the current system only provides rudimentary support for OWL-S. It should be investigated whether or not it is possible for the system to seamlessly make use of services that are written in different languages.

As for service discovery it should be investigated how the system can be integrated with remote service repositories in order to provide a broader assortment of services to the users. It should be possible to create some sort of distributed approach to service discovery. Perhaps a master category directory can be established? Such a list could have links to service repositories that contain services within a particular category.

Investigate New and Existing Frameworks

The existing Semantic Web frameworks that have been described in the prestudy report [16] and the Related Work chapter in this report should be watched in the future. There is much to learn from the experiences these projects have made and their products may also be used in future versions of the system developed in this project. Projects that are of particular interest are the WOSE [55] project, the METEOR-S [31] project and finally the WSMO [52] project.

Also eventual new frameworks and tools should be investigated as they appear. As said
in the beginning of this report there is a lot of activity in the Semantic Web community, so better and more powerful tools are bound to appear in the near future.

**Knowledge Worker Clients**

Since the system developed in this project is a back-end system, creating clients should be a priority for any eventual future work. By creating clients, for example to fulfil the scenarios listed in Section 4.2, the practical usage of the system can be evaluated. It can also be studied how the system and the Semantic Web in general can be used to improve the daily work of knowledge workers.

The first step in creating a client should be to create a Requirements Editor (RE), the RE provides the end user with a graphical user interface for creating and manipulating workflows that make use of our system. The RE should be created in a way that makes it easy to embed the RE into end user clients. The reason for this is that the functionality featured in the RE will be reusable across most end user clients.

When some actual clients are created for the system, it will be possible to test the practical value of the system, and investigate if any features are missing or useless.

**Investigate Theoretical Principles**

The theoretical aspects around the use of Semantic Web Services for improving the work situation of knowledge workers should be investigated. Such an investigation would show which principles that work in practice, and perhaps unravel new principles and usage areas for Semantic Web Service technologies. Also this investigation will show how the framework and SWSG system can be evolved in the future.

**Build a Complete Test Collection**

A complete test collection should be created for the system, e.g. a collection of working Semantic Web Services along with related queries. This collection will enable the creation of a complete test environment for the system. When a complete test collection have been established the system can be tested more thoroughly, and it will also be easier to test new components.
Chapter 18

Conclusions

This project has defined a set of goals and requirements that should be relevant for many applications that make use of Semantic Web Services. We have also identified a series of problems and possible solutions for the combination of Semantic Web tools into more user friendly and general frameworks.

During the life cycle of this project we have managed to produce a running framework for Semantic Web Service applications. While researching and implementing we have also gathered a lot of experience along the way. The framework – which is the result of this project – enable the creation of Semantic Web enabled clients, which can provide new information regarding the current usability of Semantic Web technologies for end users. It should be noted, however, that the framework in its current state does have a limitation regarding the Invocation Service. This flaw is described in Chapter 17 Future Work.

One of the main strengths of the framework is its architecture. The architecture has been created with modifiability, extendability and testability in mind. This architecture will enable the framework to be used for testing new Semantic Web technologies as they mature. Also, as mentioned above, the framework enables the creation of a wide array of exiting new end-user tools.

While developing the framework we managed to test several of the available frameworks and tools for OWL-S based Semantic Web Services. Our experiences with these tools and frameworks are described in this report.

We have achieved the concrete goals as presented in the Problem Definition (See Section 1.2). We have refined the requirements from the prestudy [16] and used these requirements to design an architectural framework for Semantic Web back-end systems. We have also satisfied the last two goals by implementing a prototype using our architecture and design that is based on existing tools for the Semantic Web.

Finally, we believe this project has provided a sturdy foundation for further research on tools for the Semantic Web. Tools that some day will help end users in their daily work.
Bibliography


[34] Nielsen//NetRatings. Internet Usage Statistics. 


Appendix
Appendix A

Glossary

Agent

Agents, are defined in [6] as software programs that gather or process information in the background. Intelligent agents are a vital part of the Semantic Web, as the features of the SW will enable them to gather and process information automatically based on a goal provided by a user [5].

Best of Breed

Best of Breed (BoB) applications, are software systems that are built by combining existing applications or modules into a complete system tailored to fit the needs of the user. Getting more widespread since the adaption of SOA is providing more modular applications that are more integration friendly than the enormous legacy systems from the past.

WfMS

WfMS is short for Workflow Management System. According to [22] a WfMS is a system that:

“Allow organizations to define and control the various activities associated with a business process. In addition, many management systems also and analyze the execution of the process so that continuous improvements can be made.”
Appendix B

Maven2 Command Reference

In Table B.1 below we have listed the some important Maven2 commands for managing our project. All commands must be given in the project root directory, that contains the POM file (pom.xml). Note that Maven must be properly installed and set up for these commands to work. For more information regarding Maven, see the project Web site\(^1\).

<table>
<thead>
<tr>
<th>Action</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build project</td>
<td>mvn compile</td>
</tr>
<tr>
<td>Run JUnit tests (results are logged to file)</td>
<td>mvn test</td>
</tr>
<tr>
<td>Parameter for skipping tests</td>
<td>mvn -Dmaven.test.skip (&lt;\text{OPERATION}&gt;)</td>
</tr>
<tr>
<td>Parameter for offline mode (use local repository)</td>
<td>mvn -Doffline=true (&lt;\text{OPERATION}&gt;)</td>
</tr>
<tr>
<td>Multiple commands can be given at once</td>
<td>mvn clean test package</td>
</tr>
<tr>
<td>Install custom package in local repository</td>
<td>mvn install:install-file -Dfile=FILENAME.jar -DgeneratePom=true -DgroupId=MY.GROUP -DartifactId=MY_ARTIFACT -Dpackaging=jar -Dversion=MY_VERSION</td>
</tr>
<tr>
<td>Create distribution</td>
<td>mvn package</td>
</tr>
<tr>
<td>Generate Javadoc</td>
<td>mvn javadoc:javadoc</td>
</tr>
<tr>
<td>Clean project (delete generated files)</td>
<td>mvn clean</td>
</tr>
<tr>
<td>Generate Eclipse project files</td>
<td>mvn eclipse:eclipse</td>
</tr>
<tr>
<td>Generate IntelliJ IDEA project files</td>
<td>mvn idea:idea</td>
</tr>
<tr>
<td>Generate Project Web site</td>
<td>mvn site:site</td>
</tr>
<tr>
<td>Deploy site to Web server (requires setup in POM)</td>
<td>mvn site:site-deploy</td>
</tr>
<tr>
<td>Deploy project to app server (requires setup in POM)</td>
<td>mvn cargo:deploy</td>
</tr>
<tr>
<td>Install current project into local repository</td>
<td>mvn install</td>
</tr>
</tbody>
</table>

Table B.1: Maven2 commands for handling the project

\(^1\)http://maven.apache.org/
Appendix C

File Attachements

This chapter describes the contents in the included zip-archive.

Documentation

For documentation the report is included in Adobe PDF format in the report directory, and the JavaDoc documentation for the source code is included under the javadoc directory as HTML. We have also included the \LaTeX source code along with the source images under the report/latex directory.

Source Code

The source code for the project is available in the source directory. It is published in the form of two Maven 2 projects, the first being the main JAR project for the prototype under the source/sws4bpe1 directory. And the second being the Web Service Remote Façade for the system under the source/sws4bpe1_web directory.

For information on using Maven 2 to manage the project, please refer to Appendix B. Also note that a prebuilt binary distribution of both components is available in the included Maven 2 repository snapshot.

Maven 2 Repository Snapshot

In the maven2 directory we have included a snapshot of our local Maven 2 repository. This contains most libraries needed to build the project.
Appendix D

Source Code

This appendix contains a complete printout of the source code for the SWSG prototype, excluding the test cases. We have also included the main configuration file (sws4bpel.properties) and the configuration files for Spring, Hibernate and the Web Application.
package no.ntnu.idi.wisemod.sws4bpel.domain;

import java.net.URI;
import java.io.Serializable;
import java.util.Vector;

public class Service implements Serializable, Comparable<Service> {

    /**
     * Location of the service as a standard URI
     */
    private URI location;

    /**
     * Ranking of the service, for internal usage
     */
    private int ranking;

    /**
     * Vector containing input URI concepts
     */
    private Vector<URI> inputURIs;

    /**
     * Vector containing output URI concepts
     */
    private Vector<URI> outputURIs;

    /**
     * Specified by serialVersionUID
     */
    public static final long serialVersionUID = 2L;

    /**
     * Default constructor
     */
    public Service() {
        this.location = null;
        this.ranking = -1;
        this.inputURIs = new Vector<URI>();
        this.outputURIs = new Vector<URI>();
    }

    /**
     * Constructor
     * @param location URI to the service
     */
    public Service(URI location) {
        this.location = location;
    }

    /**
     * Constructor
     * @param location URI to the service
     * @param id Internal ID of service
     */
    public Service(URI location, Integer id) {
        this(location);
        this.id = id;
    }

    /**
     * Constructor
     * @param location URI to the service
     * @param id Internal ID of service
     * @param inputURIs Input concept URIs
     * @param outputURIs Output Concept URIs
     */
    public Service(URI location, Integer id, Vector<URI> inputURIs, Vector<URI> outputURIs) {
        this(location);
        this.id = id;
        this.inputURIs = inputURIs;
        this.outputURIs = outputURIs;
    }

    /**
     * Set internal ID
     * @param id Internal ID
     */
    public void setId(Integer id) {
        this.id = id;
    }
}
/**
 * Set service URI
 * @param location URI of the service
 */
 public void setLocation(URI location) {
   this.location = location;
}

/**
 * Set internal service ranking (lower is better!)
 * @param ranking Internal service ranking
 */
 public void setRanking(int ranking) {
   this.ranking = ranking;
}

/**
 * Set input concept URI Vector
 * @param inputURIs Vector containing input URIs
 */
 public void setInputURIs(Vector<URI> inputURIs) {
   this.inputURIs = inputURIs;
}

/**
 * Add an input concept URI
 * @param inputURI Concept URI to add
 */
 public void addInputURI(URI inputURI) {
   this.inputURIs.add(inputURI);
}

/**
 * Set output URI vector
 * @param outputURIs Vector containing output concept URIs
 */
 public void setOutputURIs(Vector<URI> outputURIs) {
   this.outputURIs = outputURIs;
}

/**
 * Add an output concept URI
 * @param outputURI Concept URI to add
 */
 public void addOutputURI(URI outputURI) {
   this.outputURIs.add(outputURI);
}

/**
 * Get internal service ID
 * @return Internal service ID
 */
 public Integer getId() {
   return id;
}

/**
 * Get Service location URI
 * @return Service URI
 */
 public URI getLocation() {
   return location;
}

/**
 * Get internal Service ranking
 * PSI Less is better!
 * @return Ranking of the Service
 */
 public int getRanking() {
   return ranking;
}

/**
 * Get input concept URIs
 */
public Vector<URI> getInputURIs() {
    return inputURIs;
}

public Vector<URI> getOutputURIs() {
    return outputURIs;
}

@override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }
    Service that = (Service) o;
    if (!id.equals(that.id)) {
        return false;
    }
    if (!location.equals(that.location)) {
        return false;
    }
    return true;
}

@override
public int compareTo(Service service) {
    if (service.getRanking() == this.ranking) {
        return 0;
    }
    return this.ranking - service.getRanking();
}

@override
public int hashCode() {
    int result;
    result = id.hashCode();
    result = 29 * result + location.hashCode();
    return result;
}

@override
public String toString() {
    String tmp = "";
    if (this.id != null) {
        tmp += this.id + "\n";
    }
    return tmp + this.location + "\n" + this.id + "\n" + this.ranking;
}
else {
  tmp += "(NO ID), ";
}

if (this.location != null) {
  tmp += this.location.toString() + ", ";
} else {
  tmp += "(NO URI), ";
}

return tmp + this.ranking;
package no.ntnu.idi.wisemod.sws4bpel.domain;

import java.net.URI;
import java.util.HashMap;
import java.util.Vector;
import java.io.Serializable;

/**
 * This class represents a Query or user request, and contains everything needed in order to match the users request to eventual available services.
 */

// TODO: Add QoS parameters. PS! Remember to update Hibernate definition! ServiceQuery.hbm.xml

public class ServiceQuery implements Serializable {

    private Integer id;

    /** Contains the URIs for the required input concepts along with the values that are to be passed on */
    private HashMap<URI, String> inputs;

    /** Contains the URIs for the input concepts */
    private Vector<URI> outputs;

    /** URIs for the ontology imports */
    private Vector<URI> imports;

    /** Defined by @see java.io.Serializable#serialVersionUID */
    public static final long serialVersionUID = 1L;

    /**
     * Default constructor
     */
    public ServiceQuery() {
        this.id = null;
        this.inputs = new HashMap<URI, String>();
        this.outputs = new Vector<URI>();
        this.imports = new Vector<URI>();
    }

    /**
     * Constructor
     *
     * @param inputs Contains the URIs for the required input concepts along with the values that are to be passed on
     * @param outputs Contains the URIs for the input concepts
     * @param imports URIs for the ontology imports
     */
    public ServiceQuery(HashMap<URI, String> inputs, Vector<URI> outputs, Vector<URI> imports) {
        this.id = null;
        this.inputs = inputs;
        this.outputs = outputs;
        this.imports = imports;
    }

    /**
     * Set the internal identifier
     *
     * @param id Query id
     */
    public void setId(Integer id) {
        this.id = id;
    }

    /**
     * Set input HashMap
     *
     * @param inputs Contains the URIs for the required input concepts along with the values that are to be passed on
     */
    public void setInputs(HashMap<URI, String> inputs) {
        this.inputs = inputs;
    }

    /**
     * Add an input concept and an input value
     *
     * @param uri URI to the concept type
     * @param value Value to pass on to the target service
     */
    public void addInput(URI uri, String value) {
        this.inputs.put(uri, value);
    }

    /**
     * Set output Set
     *
     * @param outputs Contains the URIs for the input concepts
     */
}
public void setOutputs(Vector<URI> outputs) {
    this.outputs = outputs;
}

/**
 * Add an output concept
 * @param output URI to the output concept
 */
public void addOutput(URI output) {
    this.outputs.add(output);
}

/**
 * Set import Set
 * @param imports URIs for the ontology imports
 */
public void setImports(Vector<URI> imports) {
    this.imports = imports;
}

/**
 * Add an import URI
 * @param importURI URI to the import to be added
 */
public void addImport(URI importURI) {
    this.imports.add(importURI);
}

/**
 * Get the internal identifier
 * @return Internal Query id
 */
public Integer getId() {
    return id;
}

/**
 * Return Input URIs and values
 * @return HashMap containing the URIs for the required input concepts along with the values that are to be passed on
 */
public HashMap<URI, String> getInputs() {
    return inputs;
}

/**
 * Return a Set containing all output URIs
 * @return Set containing the URIs for the input concepts
 */
public Vector<URI> getOutputs() {
    return outputs;
}

/**
 * Return a Set containing all imports
 * @return Set containing URIs for the ontology imports
 */
public Vector<URI> getImports() {
    return imports;
}

/**
 * Defined by (@see java.lang.Object#equals)
 * @param o Object to compare an instance to
 * @return <code>true</code> if the objects are equal or <code>false</code> otherwise
 */
@EqualsAndHashCode
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    ServiceQuery that = (ServiceQuery) o;

    if (!imports.equals(that.imports)) return false;
    if (!inputs.equals(that.inputs)) return false;
    if (!outputs.equals(that.outputs)) return false;

    return true;
}
return true;
}

/**
 * Defined by {@see java.lang.Object#hashCode}
 */
@override
public int hashCode() {
    int result;
    result = inputs.hashCode();
    result = 29 * result + outputs.hashCode();
    result = 29 * result + imports.hashCode();
    return result;
}

/**
 * Specified by {@see java.lang.Object#toString}
 */
@override
public String toString() {
    String tmp = "ServiceQuery:
    if (this.inputs != null && this.inputs.size() != 0) {
        tmp += "Inputs:
        for (URI uri : this.inputs.keySet()) {
            tmp += uri.toString() + " -> " + this.inputs.get(uri) + "\n";
        }
    } else {
        tmp += "(NO INPUTS)\n";
    }

    if (this.outputs != null && this.outputs.size() != 0) {
        tmp += "Outputs:\n        for (URI uri : this.outputs) {
            tmp += uri.toString() + "\n";
        }
    } else {
        tmp += "(NO OUTPUTS)\n";
    }

    if (this.imports != null && this.imports.size() != 0) {
        tmp += "Imports:\n        for (URI uri : this.imports) {
            tmp += uri.toString() + "\n";
        }
    } else {
        tmp += "(NO IMPORTS)\n";
    }
    return tmp;
}
package no.ntnu.idi.wisemod.sws4bpel.domain;

import java.io.Serializable;
import java.net.URI;
import java.util.HashMap;

/**
 * Holds the result of an invoked service
 * Could really just have used a (\see java.util.HashMap) here...
 * But this is more semantic and better for eventual future additions
 */

public class ServiceResult implements Serializable {

    /**
     * HashMap holding the result value (String) for each concept (URI)
     */
    private HashMap<URI,String> results;

    public static final long serialVersionUID = 1L;

    /**
     * Default constructor
     */
    public ServiceResult() {
        this.results = new HashMap<URI,String>();
    }

    /**
     * Constructor
     * @param results HashMap containing the concept URI's and their respective values
     */
    public ServiceResult(HashMap<URI,String> results) {
        this.results = results;
    }

    /**
     * Set resultmap
     * @param results HashMap containing the concept URI's and their respective values
     */
    public void setResults(HashMap<URI,String> results) {
        this.results = results;
    }

    /**
     * Add a single result
     * @param uri URI of the output concept
     * @param value value of the field
     */
    public void addResult(URI uri, String value) {
        this.results.put(uri,value);
    }

    /**
     * Return all results in a HashMap
     * @return HashMap containing the concept URI's and their respective values
     */
    public HashMap<URI,String> getResults() {
        return results;
    }

    /**
     * Defined by (\see java.lang.Object.equals)
     * @param o Object to compare an instance to
     * @return <code>true</code> if the objects are equal or <code>false</code> otherwise
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        final ServiceResult that = (ServiceResult) o;

        if (!results.equals(that.results)) return false;

        return true;
    }

    /**
     * Specified by (\see java.lang.Object#hashCode)
     */
    public int hashCode() {
        return 0;
    }
}
```java
 *@return hashcode
 */

@Override
public int hashCode() {
    return results.hashCode();
}

/**
 * Specified by (see java.lang.Object#toString)
 *
 */

@Override
public String toString() {
    String tmp = "ServiceResult:
    if (this.results != null && this.results.size() != 0) {
        for (URI uri : this.results.keySet()) {
            tmp += uri.toString() + " -> " + this.results.get(uri) + "\n"
        }
    } else {
        tmp = "(NO RESULTS)\n"
    }
    return tmp;
}
```
package no.ntnu idi.wisemod.sws4bpel.domain;

import java.io.Serializable;
import java.util.*;

/**
 * Represents a stored User Session
 */
public class Session implements Serializable{

    // TODO: An incremental integer id might not be the best choice here..
    /** Unique identifier for this session */
    private Integer id;
    /** The original client query */
    private ServiceQuery query;
    /** Set containing all matched services */
    private Set<Service> matchedServices;

    /**
     * Default constructor
     */
    public Session() {
        this.id = -1;
        this.query = null;
        this.matchedServices = new TreeSet<Service>();
    }

    /**
     * Constructor
     * @param id Unique identifier for this session
     * @param query Service Query from the client's request
     * @param matchedServices matched services from the matchmaker
     */
    public Session(Integer id, ServiceQuery query, Set<Service> matchedServices) {
        this.id = id;
        this.query = query;
        this.matchedServices = matchedServices;
    }

    /**
     * Constructor
     * @param query Service Query from the client's request
     */
    public Session(ServiceQuery query) {
        this(null, query, new TreeSet<Service>());
    }

    /**
     * Set id
     * @param id unique identifier for the session
     */
    public void setId(Integer id) {
        this.id = id;
    }

    /**
     * Set query
     * @param query client's query object
     */
    public void setQuery(ServiceQuery query) {
        this.query = query;
    }

    /**
     * Set matched services
     * @param matchedServices set containing matched services from the matchmaker
     */
    public void setMatchedServices(Set<Service> matchedServices) {
        this.matchedServices = matchedServices;
    }

    /**
     * Add a matched service
     * @param matchedService
     */
    public void addMatchedService(Service matchedService) {
        this.matchedServices.add(matchedService);
    }
/**
 * Add matched services from a Vector
 * @param matchedServices Vector to add services from
 */
 public void addMatchedServices(Vector<Service> matchedServices) {
   this.matchedServices.addAll(matchedServices);
 }

/**
 * @return unique identifier for the session
 */
 public Integer getId() {
   return id;
 }

/**
 * @return client's query object
 */
 public ServiceQuery getQuery() {
   return query;
 }

/**
 * @return set containing matched services from the matchmaker
 */
 public Set<Service> getMatchedServices() {
   return matchedServices;
 }

/**
 * Get matched service by id
 * @param serviceld ID of the service to find
 * @return Service with requested id, or <code>null</code> if none is found
 */
 public Service getMatchedServiceById(Integer serviceld) {
   for (Service s : this.matchedServices) {
     if (s.getId() == serviceld) {
       return s;
     }
   }
   return null;
 }

/**
 * Specified by {@code @see java.lang.Object.equals}
 * @param o Object to compare this instance with
 * @return <code>true</code> if the objects are equal, else <code>false</code>
 */
 @Override
 public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass()) return false;
   final Session session = (Session) o;
   if (!ld.equals(session.id)) return false;
   if (!query.equals(session.query)) return false;
   return true;
 }

/**
 * Specified by {@code @see java.lang.Object#hashCode}
 * @return hashcode of this instance
 */
 @Override
 public int hashCode() {
   int result;
   result = id.hashCode();
   result = 29 * result + query.hashCode();
   return result;
 }
/**
 * Specified by @see java.lang.Object#toString
 * @return String representation of this object, for debugging purposes..
 */

@Override
public String toString() {
    String tmp = "Session " + this.id + " for query:\n"
    if (this.query != null) {
        tmp += this.query.toString();
    } else {
        tmp += "(NO QUERY)";
    }
    tmp += "\nMatched Services:\n"
    if (this.matchedServices != null && this.matchedServices.size() != 0) {
        for (Service s : this.matchedServices) {
            tmp += s.toString() + "\n";
        }
    } else {
        tmp += "(NO MATCHED SERVICES)";
    }
    return tmp;
}
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="no.ntnu.idi.wisemod.sws4bpel.domain">
  <class name="ServiceQuery" table="SERVICEQUERY">
    <id name="id" column="SERVICEQUERY_ID" type="java.lang.Integer">
      <generator class="increment"/>
    </id>
    <property name="inputs" type="java.util.HashMap" column="INPUTS" not-null="true"/>
    <property name="outputs" type="java.util.Vector" column="OUTPUTS" not-null="true"/>
    <property name="imports" type="java.util.Vector" column="IMPORTS" not-null="true"/>
  </class>
</hibernate-mapping>
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="no.ntnu.idi.wisemod.sws4bpel.domain">
  <class name="Service" table="SERVICE">
    <id name="id" column="SERVICE_ID" type="java.lang.Integer">
      <generator class="increment"/>
    </id>
    <property name="location" type="java.net.URI" column="LOCATION" not-null="true"/>
  </class>
</hibernate-mapping>
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="no.ntnu.idi.wisemod.sws4bpel.domain">  
<class name="Session" table="SESSION">  
   <id name="id" column="SESSION_ID" type="java.lang.Integer">  
    <generator class="increment"/>  
   </id>  
   <!-- one-to-one name="query" class="ServiceQuery" -->  
   <!-- TODO: really one-to-one... but it did not work,  
   probably because of the missing insert property?  
   or perhaps a formula should be added? no time to investigate  
   -->  
   <many-to-one name="query" class="ServiceQuery"  
    column="SERVICEQUERY_ID"  
    not-null="true"  
    update="true"  
    insert="true"  
    cascade="all"  
    foreign-key="SERVICEQUERY_ID"  
    lazy="false"/>  
   <set name="matchedServices" table="SERVICE_SESSION" cascade="none" lazy="false">  
    <key column="SERVICE_ID"/>  
    <many-to-many column="SESSION_ID" class="Service"/>  
   </set>  
</class>  
</hibernate-mapping>
package no.ntnu.idi.wisemod.sws4bpel.registry;

/**
 * Generic exception class for the Registry Service package
 */
public class RegistryException extends Exception {
    public RegistryException() {
    }
    public RegistryException(String string) {
        super(string);
    }
    public RegistryException(String string, Throwable throwable) {
        super(string, throwable);
    }
    public RegistryException(Throwable throwable) {
        super(throwable);
    }
}
package no.ntnu.idi.wisemod.sws4bpel.registry;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.Session;
import java.util.List;
import java.net.URI;

/**
 * Interface defining the services of the RegistryService
 */
public interface RegistryService {

  public Service registerService(Service service) throws RegistryException;
  public void removeService(Integer serviceId) throws RegistryException;
  public Service findServiceById(Integer serviceId);
  public Service findServiceByURI(URI uri);
  public List<Service> findAllServices();
  public Session registerSession(Session session) throws RegistryException;
  public void removeSession(Integer sessionId) throws RegistryException;
  public Session findSessionById(Integer sessionId);
}
package no.ntnu.idi.wisemod.sws4bpel.registry;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.Session;
import no.ntnu.idi.wisemod.sws4bpel.registry.dao.ServiceDAO;
import no.ntnu.idi.wisemod.sws4bpel.registry.dao.SessionDAO;
import java.util.List;
import java.net.URI;

/**
 * Implementation of the Registry Service interface
 */
public class RegistryServiceImpl implements RegistryService {

    /**
     * Instance of ServiceDAO *
     */
    private ServiceDAO serviceDAO;

    /**
     * Instance of SessionDAO *
     */
    private SessionDAO sessionDAO;

    /**
     * Set the ServiceDAO instance
     */
    @param serviceDAO Instance of ServiceDAO
    */
    public void setServiceDAO(ServiceDAO serviceDAO) {
        this.serviceDAO = serviceDAO;
    }

    /**
     * Set the SessionDAO instance
     */
    @param sessionDAO Instance of SessionDAO
    */
    public void setSessionDAO(SessionDAO sessionDAO) {
        this.sessionDAO = sessionDAO;
    }

    /**
     * Get the ServiceDAO instance
     */
    @return serviceDAO
    */
    public ServiceDAO getServiceDAO() {
        return serviceDAO;
    }

    /**
     * Get the SessionDAO instance
     */
    @return session dao
    */
    public SessionDAO getSessionDAO() {
        return sessionDAO;
    }

    /**
     * Register a service in the persistent data store
     */
    @param service Service to persist
    @return Persist a service, with ia set
    @throws RegistryException if service is already registered
    */
    public Service registerService(Service service) throws RegistryException {
        Service existing = this.serviceDAO.findByURI(service.getLocation()); 
        if (existing == null) {
            return this.serviceDAO.save(service);
        } else {
            throw new RegistryException("Service is already registered with id " + existing.getId());
        }
    }

    /**
     * Remove a persistent service
     */
    @param serviceId Id of the service to remove
    @throws RegistryException on error
    */
    public void removeService(Integer serviceId) throws RegistryException {
        if (!this.serviceDAO.isRegistered(serviceId)) {
            throw new RegistryException("Service with id " + serviceId + " does not exist");
        }
    }
}
/**
 * Find a persistent service by identifier
 * @param serviceId ID to find service by
 * @return persisted service, or <code>null</code> if none is found
 */
public Service findServiceById(Integer serviceId){
    return this.serviceDAO.findById(serviceId);
}

/**
 * Find all persisted services
 * @return list containing all services from the persistent data store
 */
public List<Service> findAllServices() {
    return this.serviceDAO.findAll();
}

/**
 * Find persisted service by URI
 * @param uri URI of service to find
 * @return persisted service or <code>null</code> if none is found
 */
public Service findServiceByURI(URI uri) {
    return this.serviceDAO.findByURI(uri);
}

// TODO: These methods should be secured, since it is exposed to all clients through the ServiceLocator in the Facade

/**
 * Persist a user session
 * @param session The Session instance to persist
 * @return Session instance with persistence id
 * @throws RegistryException on error
 */
public Session registerSession(Session session) throws RegistryException {
    try {
        return this.sessionDAO.save(session);
    } catch (RuntimeException e) { // catch RuntimeException instead of DataAccessException which is Spring specific
        throw new RegistryException("Could not persist session", e);
    }
}

/**
 * Delete a user session from the persistence store
 * @param sessionId identifier of the session to delete
 * @throws RegistryException on error
 */
public void removeSession(Integer sessionId) throws RegistryException {
    try {
        this.sessionDAO.delete(sessionId);
    } catch (RuntimeException e) { // catch (RuntimeException e) {
        throw new RegistryException("Could not remove persisted session with id: "+sessionId,e);
    }
}

/**
 * Return a persisted session by id
 * @param sessionId the identifier of the persisted session
 * @return Persisted session instance or <code>null</code> if none is found
 */
public Session findSessionBySessionId(Integer sessionId) {
    return this.sessionDAO.findBySessionId(sessionId);
}
package no.ntnu.idi.wisemod.sws4bpel.registry.dao;

import java.util.List;
import java.io.Serializable;

/**
 * Generic DAO interface, should be extended by all DAO interfaces.
 * This is an implementation of the Generic DAO Pattern for Java 5, as proposed by the Hibernate team:
 */

public interface GenericDAO<T, ID extends Serializable> {

    /**
     * Find a persistent entity by it's identifier
     * @param id ID of the persistent entity
     * @param lock Should the entity be locked for read/write?
     * @return persistent entity or <code>null</code> if not found
     */
    public T findById(ID id, boolean lock);

    /**
     * Find all persistent entities of this type
     * @return list containing all entities in persistent store
     */
    public List<T> findAll();

    /**
     * Find a persistent entity by an example instance
     * @param exampleInstance example instance to search for
     * @param excludeProperties fields that should be ignored in the search
     * @return matched instance
     */
    public List<T> findByExample(T exampleInstance, String ... excludeProperties);

    /**
     * Save an entity to the persistent store
     * @param entity to persist
     * @return stored entity (with id set)
     */
    public T save(T entity);

    /**
     * Delete an entity from the persistent store
     * @param entity Entity to delete
     */
    public void delete(T entity);

    /**
     * Delete an entity by it's ID
     * @param id ID of the entity to delete
     */
    public void delete(ID id);

    /**
     * Close all connections and clean up
     */
    public void close();
}
package no.ntnu.idi.wisemod.sws4bpel.registry.dao;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import java.net.URI;

/**
 * Interface, defining the DAO functionality for the @see no.ntnu.idi.wisemod.sws4bpel.domain.Service domain class
 */
public interface ServiceDAO extends GenericDAO<Service, Integer> {

    /**
     * Find a service in the datasource by URI
     * @param uri Unique URI for the service
     * @return Service with the requested URI or <code>null</code> if not found
     */
    public Service findByURI(URI uri);

    /**
     * Find out if the service with given URI is already registered
     * @param uri URI to check
     * @return <code>true</code> if the URI is already registered, else <code>false</code>
     */
    public boolean isRegistered(URI uri);

    /**
     * Find out if the service with given identifier is already registered
     * @param serviceId ID to check
     * @return <code>true</code> if the ID is already registered, else <code>false</code>
     */
    public boolean isRegistered(Integer serviceId);
}
package no.ntnu.idi.wisemod.sws4bpel.registry.dao;

import no.ntnu.idi.wisemod.sws4bpel.domain.Session;

/**
 * Interface, defining the DAO functionality for the @see no.ntnu.idi.wisemod.sws4bpel.domain.Session domain class
 */
public interface SessionDAO extends GenericDAO<Session,Integer> {
package no.ntnu.idi.wisemod.sws4bpel.registry.dao.hibernate;

import org.hibernate.criterion.Example;
import org.hibernate.*;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import org.springframework.orm.hibernate3.HibernateCallback;
import org.springframework.dao.DataAccessException;

import java.io.Serializable;
import java.lang.reflect.ParameterizedType;
import java.util.List;
import java.util.List;

import no.ntnu.idi.wisemod.sws4bpel.registry.dao.GenericDAO;

/**
 * Implementation of the Generic DAC interface for Hibernate 3 (using Spring Hibernate Support)
 */

public abstract class GenericHibernateDAO<T, ID extends Serializable, DAOIF extends GenericDAO<T,ID>> extends HibernateDaoSupport implements GenericDAO<T,ID> {

/**
 * The class type of the entity we wish to persist */
 private Class<T> persistentClass;

/**
 * Default constructor
 */

public GenericHibernateDAO() {

 this.persistentClass =
 (Class<T>) ((ParameterizedType) getClass().getGenericSuperclass()).getActualTypeArguments()[0];
}

/**
 * Return instance of the DAO
 */
 @param sessionFactory Hibernate session factory
 @return Initialize instance of DAO
 @throws DataAccessException
 */
 @SuppressWarnings("unchecked")
 public DAOIF getInstance(SessionFactory sessionFactory) throws DataAccessException {

 this.setSessionFactory(sessionFactory);

 return (DAOIF) this;
 }

/**
 * Return the class with this DAO persists
 */
 @return Class of the entity this DAO manages
 */
 protected Class<T> getPersistentClass() {

 return this.persistentClass;
 }

/**
 * Find a persistent entity by it's identidier
 */
 @param id ID of the persistent entity
 @return persistent entity or <code>null</code> if not found
 @throws DataAccessException on error (unchecked runtime exception)
 */
 public T findById(ID id) throws DataAccessException {

 return this.findById(id, false);
 }

/**
 * Find a persistent entity by it's identidier
 */
 @param id ID of the persistent entity
 @param lock Should the entity be locked for read/write?
 @return persistent entity or <code>null</code> if not found
 @throws DataAccessException on error (unchecked runtime exception)
 */
 @SuppressWarnings("unchecked")
 public T findById(ID id, boolean lock) throws DataAccessException {

 T instance;
 if (lock) {

  instance = (T) this.getHibernateTemplate().get(this.getPersistentClass(), id, LockMode.UPGRADE);
 } else {

  instance = (T) this.getHibernateTemplate().get(this.getPersistentClass(), id);
 }

*/
public List<T> findExampleInstance(Final T exampleInstance, final String ... excludeProperties)
throws DataAccessException {
    return (List<T>) this.getHibernateTemplate().execute(
            new HibernateCallback() { 
            public Object doInHibernate(Session session) throws HibernateException {
                Criteria criteria = session.createCriteria(getPersistentClass());
                Example example = Example.create(exampleInstance);
                for (String exclude : excludeProperties) {
                    example.excludeProperty(exclude);
                }
                return criteria.list();
            });
        }

    };
}

/**
* Save an entity to the persistent store
* @param instance instance to persist
* @return stored entity (with id set)
* @throws DataAccessException on error (unchecked runtime exception)
*/
public T save(T instance) throws DataAccessException {
    this.getHibernateTemplate().saveOrUpdate(instance);
    return instance;
}

/**
* Delete an entity from the persistent store
* @param instance instance to delete
* @throws DataAccessException on error (unchecked runtime exception)
*/
public void delete(T instance) throws DataAccessException {
    this.getHibernateTemplate().delete(instance);
}

/**
* Delete an entity by it's ID
* @param id ID of the entity to delete
* @throws DataAccessException on error (unchecked runtime exception)
*/
public void delete(ID id) throws DataAccessException {
    T toDelete = this.findById(id, true);
    this.getHibernateTemplate().delete(toDelete);
}

/**
* Close all connections and clean up
* @throws DataAccessException on error (unchecked runtime exception)
*/
public void close() throws DataAccessException {
    this.getHibernateTemplate().flush();
}
155
156  }
157
package no.ntnu.idi.wisemod.sws4bpel.registry.dao.hibernate;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.registry.dao.ServiceDAO;
import java.net.URI;
import java.util.List;

/**
 * Hibernate implementation of the Service DAO interface
 */
public class ServiceHibernateDAO extends GenericHibernateDAO<Service, Integer, ServiceDAO>
implements ServiceDAO {

    /**
     * Find a service in the datasource by URI
     * @param uri Unique URI for the service
     * @return Service with the requested URI or null if not found
     */
    public Service findByURI(URI uri) {
        Service example = new Service(uri);
        List<Service> list = this.findByIdExample(example, "id", "ranking");
        if (list.size() > 0) {
            return list.get(0);
        }
        return null;
    }

    /**
     * Find out if the service with given URI is already registered
     * @param uri URI to check
     * @return true if the URI is already registered, else false
     */
    public boolean isRegistered(URI uri) {
        if (this.findByIdExample(uri) == null) {
            return false;
        }
        return true;
    }

    /**
     * Find out if the service with given identifier is already registered
     * @param serviceId ID to check
     * @return true if the ID is already registered, else false
     */
    public boolean isRegistered(Integer serviceId) {
        if (this.findByIdExample(serviceId) == null) {
            return false;
        }
        return true;
    }
}
package no.ntnu.idi.wisemod.sws4bpel.registry.dao.hibernate;

import no.ntnu.idi.wisemod.sws4bpel.registry.dao.SessionDAO;
import no.ntnu.idi.wisemod.sws4bpel.domain.Session;

/**
 * Hibernate implementation for the Session DAO interface
 */
public class SessionHibernateDAO extends GenericHibernateDAO<
    Session, Integer, SessionDAO> implements SessionDAO {

}
package no.ntnu.idi.wisemod.sws4bpel.core;

/**< Generic unchecked exception class for the SWS4BPEL Facade */
public class S4BGenericException extends RuntimeException {
    public S4BGenericException() {
    }
    public S4BGenericException(String string) {
        super(string);
    }
    public S4BGenericException(String string, Throwable throwable) {
        super(string, throwable);
    }
    public S4BGenericException(Throwable throwable) {
        super(throwable);
    }
}
package no.ntnu.idi.wisemod.sws4bpel.core;

import no.ntnu.idi.wisemod.sws4bpel.invocation.InvocationService;
import no.ntnu.idi.wisemod.sws4bpel.matchmaking.MatchmakerService;
import no.ntnu.idi.wisemod.sws4bpel.registry.RegistryService;
import no.ntnu.idi.wisemod.sws4bpel.selection.SelectionService;

/**
 * Interface for the ServiceLocator
 */
public interface ServiceLocator {
    public InvocationService getInvocationService();
    public MatchmakerService getMatchmakerService();
    public RegistryService getRegistryService();
    public SelectionService getSelectionService();
}
package no.ntnu.idi.wisemod.sws4bpel.core;

/**
 * Generic service lookup exception
 */
public class ServiceLocatorException extends Exception {
    public ServiceLocatorException() {
    }
    public ServiceLocatorException(String string) {
        super(string);
    }
    public ServiceLocatorException(String string, Throwable throwable) {
        super(string, throwable);
    }
    public ServiceLocatorException(Throwable throwable) {
        super(throwable);
    }
}
package no.ntnu.idi.wisemod.sws4bpel.core;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceResult;
import no.ntnu.idi.wisemod.sws4bpel.domain.Session;

import java.util.Vector;

/**
 * Interface for the top level Service Layer, this interface will be the facade that external clients work against.
 */
public interface SWS4BPELService {

/**
 * Register a service in the matchmaking and the service registry
 * @param service The service to register
 * @return The registered service (with generated internal ID)
 * @throws S4BGenericException if the service cannot be registered (Runtime exception)
 */
public Service registerService(Service service);

/**
 * Return service locator
 * @return Spring implemented service locator
 */
public ServiceLocator getServiceLocator();

/**
 * Remove a registered service
 * @param service The internal id of the service to remove
 * @throws S4BGenericException if the service cannot be removed (invalid id, etc) (Runtime exception)
 */
public void removeService(Integer serviceId);

/**
 * Perform a match using matchmaking
 * @param query The client's service query
 * @return Vector containing the results of the matching (empty vector if none are found)
 * @throws S4BGenericException On invalid query or some other matchmaking error (Runtime exception)
 */
public Vector<Service> performMatch(ServiceQuery query);

/**
 * Performs a complete matching, automatic selection and invocation process and returns the results
 * @param query Client's service query
 * @return ServiceResult containing the results of the invocation or <code>null</code> if no match could be found
 * @throws S4BGenericException if an error occurs during the process (Runtime exception)
 */
public ServiceResult performAutomaticProcess(ServiceQuery query);

/**
 * Starts a manual SWS4BPEL process. Performs matching and then stores a user session in the persistent store. This session can be resumed when the user/client has manually selected the service that will be invoked.
 * @param query Client's service query
 * @return Identifier of the stored client session (must be stored in the client)
 * @throws S4BGenericException if the invocation results in an error (Runtime exception)
 */
public Integer performManualProcess(ServiceQuery query);

/**
 * Resumes a stored session.
 * @param sessionld Identifier of the stored session
 * @param service The service that will be executed
 * @throws S4BGenericException if the invocation results in an error (Runtime exception)
 */
public ServiceResult resumeManualProcess(Integer sessionId, Integer selectedService);

/**
 * Performs an invocation on a selected service
 * @param query Client's service query (containing the input values, if any)
 * @param service The service that will be executed
 * @return ServiceResult containing the results of the invocation or <code>null</code> if no result could be found
 * @throws S4BGenericException if the invocation results in an error (Runtime exception)
 */
public ServiceResult performInvocation(ServiceQuery query, Service service);
/*
 * Performs a service selection based on system and client preferences.
 * If no preferences are given, the method returns the first of the highest ranking services as found
 * by the matchmaker.
 * @param query Client's service query (containing preferences)
 * @param matchedServices Vector containing the matched services from the matchmaker
 * @return The selected service
 * @throws S4BGenericException if something goes wrong in the selection process (Runtime exception)
 */
public Service performServiceSelection(ServiceQuery query, Vector<Service> matchedServices);
/*
 * Lookup and return session with given id
 * @param sessionId Identifier of the Session to find
 * @return Session with given id
 * @throws S4BGenericException if the session is not found (Runtime exception)
 */
public Session performSessionLookup(Integer sessionId);
package no.ntnu.idi.wisemod.sws4bpel.core;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceResult;
import no.ntnu.idi.wisemod.sws4bpel.domain.Session;
import no.ntnu.idi.wisemod.sws4bpel.registry.RegistryService;
import no.ntnu.idi.wisemod.sws4bpel.registry.RegistryException;
import no.ntnu.idi.wisemod.sws4bpel.matchmaking.MatchmakerService;
import no.ntnu.idi.wisemod.sws4bpel.matchmaking.MatchmakerException;
import no.ntnu.idi.wisemod.sws4bpel.Invocation.InvocationService;
import no.ntnu.idi.wisemod.sws4bpel.Invocation.InvocationException;
import no.ntnu.idi.wisemod.sws4bpel.selection.SelectionService;

import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.apache.log4j.Logger;
import java.util.Vector;

/**
 * This class has two roles; the first is to initialize the Spring ApplicationContext an the ServiceLocator.
 * The second is to expose the collective functionality of the systems' services to clients.
 * I.e. clients should only need to interact with this class... or a RemoteFacade using this class.
 */

public class SWS4BPELServicesImpl implements SWS4BPEServices {
    private static transient final Logger log = Logger.getLogger(no.ntnu.idi.wisemod.sws4bpel.core.SWS4BPELServicesImpl.class);

    private ServiceLocator serviceLocator;
    private RegistryService registryService;
    private MatchmakerService matchmakerService;
    private InvocationService invocationService;
    private SelectionService selectionService;

    /** Default constructor */
    public SWS4BPELServicesImpl() {
        this.serviceLocator = (ServiceLocator) appContext.getBean("serviceLocator");
        this.registryService = this.serviceLocator.getRegistryService();
        this.matchmakerService = this.serviceLocator.getMatchmakerService();
        this.invocationService = this.serviceLocator.getInvocationService();
        this.selectionService = this.serviceLocator.getSelectionService();
    }

    /**
     * Set service locator
     * @param serviceLocator Spring implemented service locator
     */
    public void setServiceLocator(ServiceLocator serviceLocator) {
        this.serviceLocator = serviceLocator;
    }

    /**
     * Return service locator
     * @return Spring implemented service locator
     */
    public ServiceLocator getServiceLocator() {
        return serviceLocator;
    }

    /**
     * Register a service in the matchmaker and the service registry
     * @param service The service to register
     */
    @param service The registered service (with generated internal ID)
    @throws S4BGenericException if service cannot be registered (Runtime exception)
public Service registerService(Service service) {
    try {
        Service registered = this.registryService.registerService(service);
        this.matchmakerService.registerService(registered);
        return registered;
    } catch (RegistryException e) {
        log.error("Could not register service", e);
        throw new S4BGenericException("Could not register service", e);
    }
}

public void removeService(Integer serviceId) {
    try {
        this.registryService.removeService(serviceId);
    } catch (RegistryException e) {
        log.error("Could not remove service", e);
        throw new S4BGenericException("Could not remove service", e);
    }
}

public Vector<Service> performMatch(ServiceQuery query) {
    try {
        return this.matchmakerService.performMatch(query);
    } catch (MatchmakerException e) {
        log.error("Could not perform service match", e);
        throw new S4BGenericException("Could not perform service match", e);
    }
}

public ServiceResult performAutomaticProcess(ServiceQuery query) {
    Vector<Service> matchedServices = this.performMatch(query);
    Service selectedService = null;
    if (matchedServices.size() != 0) {
        selectedService = this.performServiceSelection(query, matchedServices);
    } else {
        return null;
    }
    if (selectedService != null) {
        return this.performInvocation(query, selectedService);
    }
    return null;
}

// TODO: Return Session object instead of just the ID?

public Integer performManualProcess(ServiceQuery query) {
    Vector<Service> matchedServices = this.performMatch(query);

```java
155     Session userSession = new Session(query);
156     userSession.addMatchedServices(matchedServices);
157     try {
158         userSession = this.registryService.registerSession(userSession);
159         return userSession.getId();
160     } catch (RegistryException e) {
161         throw new S4BGenericException("Could not store session".e);
162     }
163 }
164 // TODO: This method should be secured, so that only the client that starts a session can complete the same session
165 // TODO: The selectedService id might be changed to a prioritized list (fallback if the selected is not available)
166 
167 /**
168 * Resumes a storea session.
169 * @param sessionId Identifier of the storea session
170 * @param selectedServiceld Identifier of the selected service
171 * @return ServiceResult containing the results of the invocation or <code>null</code> if no match could be found
172 * @throws S4BGenericException if the invocation results in an error (Runtime exception)
173 */
174 public ServiceResult resumeManualProcess(Integer sessionId, Integer selectedServiceld) {
175     Session resumedSession = this.performSessionLookup(sessionId);
176     Service selectedService = resumedSession.getMatchedServiceById(selectedServiceld);
177     if (selectedService == null) {
178         // try to fetch service from persistent store as a backup solution (the id is the same)
179         selectedService = this.registryService.findServiceById(selectedServiceld);
180         if (selectedService == null) {
181             log.warn("Could not find service with id "+selectedServiceld);
182             throw new S4BGenericException("Invalid service id "+selectedServiceld+", service could not be found")
183         }
184     }
185     ServiceQuery query = resumedSession.getQuery();
186     return this.performInvocation(query, selectedService);
187 }
188
189 /**
190 * Lookup and return session with given id
191 * @param sessionId Identifier of the Session to find
192 * @return Session with given id
193 * @throws S4BGenericException if the session is not found (Runtime exception)
194 */
195 public Session performSessionLookup(Integer sessionId) {
196     Session session = this.registryService.findSessionById(sessionId);
197     if (session == null) {
198         log.warn("Could not find session with id "+sessionId);
199         throw new S4BGenericException("Could not restore session with id "+sessionId);
200     }
201     return session;
202 }
203
204 /**
205 * Performs an invocation on a selected service
206 * @param query Client's service query (containing the input values, if any)
207 * @param service The service that will be executed
208 * @return ServiceResult or <code>null</code> if no result could be found
209 * @throws S4BGenericException if the invocation results in an error (Runtime exception)
210 */
211 public ServiceResult performInvocation(ServiceQuery query, Service service) {
212     try {
213         return this.invocationService.invoke(query, service);
214     } catch (InvocationException e) {
215         log.warn("Could not invoke service: "+service.getLocation(), e);
216         throw new S4BGenericException("Could not invoke service: "+service.getLocation(), e);
217     }
218 }
219 //**
220 */
221 // Performs a service selection based on system and client preferences.
222 // If no preferences are given, the method returns the first of the highest ranking services as found
223 // by the matchmaker.
224 } @param query Client's service query (containing preferences)
225 * @param matchedServices Vector containing the matched services from the matchmaker
226 */
```

public Service performServiceSelection(ServiceQuery query, Vector<Service> matchedServices) {
    try {
        return this.selectionService.selectService(query, matchedServices);
    } catch (Exception e) {
        throw new S4BGenericException("Could not select service", e);
    }
}

/*
 * Performs a service ranking based on system and client preferences.
 * If no preferences are given, the method sorts by the matchmaker results (Service.ranking)
 * @param query Client's service query (containing preferences)
 * @param matchedServices Vector containing the matched services from the matchmaker
 * @return Sorted Vector containing ranked services, the best matches are first
 * @throws S4BGenericException if something goes wrong in the selection process (Runtime exception)
 */
public Vector<Service> performServiceRanking(ServiceQuery query, Vector<Service> matchedServices) {
    try {
        return this.selectionService.rankServices(query, matchedServices);
    } catch (Exception e) {
        throw new S4BGenericException("Could not rank services", e);
    }
}
package no.ntnu.idi.wisemod.sws4bpel.matchmaking;

/**
 * General Exception class for the Matchmaker package.
 */
public class MatchmakerException extends Exception {
    public static final long serialVersionUID = 1L;

    public MatchmakerException()
    {
        super();
    }

    public MatchmakerException(String string)
    {
        super(string);
    }

    public MatchmakerException(String string, Throwable throwable)
    {
        super(string, throwable);
    }

    public MatchmakerException(Throwable throwable)
    {
        super(throwable);
    }
}
package no.ntnu idi.wisemod.sws4bpel.matchmaking;

import no.ntnu idi.wisemod.sws4bpel.domain.ServiceQuery;
import no.ntnu idi.wisemod.sws4bpel.domain.Service;
import java.util.Vector;
import java.io.File;

/**
 * Interface for the Matchmaker Service.
 * Defines the methods that must be implemented for each matchmaker adapter.
 */
public interface MatchmakerService {

    /**
     * Performs a match based on the provided query.
     * The result is an ArrayList containing Service instances.
     *
     * @param query Client Service Query
     * @return ArrayList containing Service instances
     * @throws MatchmakerException on a matchmaker or transformation error
     */
    public Vector<Service> performMatch(ServiceQuery query) throws MatchmakerException;

    /**
     * Registers a service with the matchmaker
     * @param service The service to register
     */
    public void registerService(Service service);

    /**
     * Transform the generic ServiceQuery instance to the native matchmaker format
     * @param query The query that is to be transformed
     * @return Transformed query as a File resource
     * @throws MatchmakerException if an error occurs during transformation
     */
    public File transformQuery(ServiceQuery query) throws MatchmakerException;
}
package no.ntnu.idi.wisemod.sws4bpel.matchmaking;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;
import no.ntnu.idi.wisemod.sws4bpel.registry.RegistryService;

import java.util.Set;
import java.util.Vector;
import java.io.IOException;
import java.io.File;
import java.io.FileWriter;
import java.net.URI;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.FileInputStream;
import java.util.Properties;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;
import java.lang.reflect.Method;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Parameter;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.TypeName;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.MethodParameter;
import java.lang.reflect.TypeName;
import java.lang.reflect.TypeVariable;
import java.lang.reflect.Type;
import java.lang.reflect.WildcardType;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.lang.reflect.AnnotatedElement;
```java
public void setLoadOnStartup(boolean loadOnStartup) {
    this.loadOnStartup = loadOnStartup;
}

/**
 * Return the OWLS-MX Matchmaker implementation
 */
public OWLS-MX Matchmaker getMatchmaker() {
    return matchmaker;
}

/**
 * Return the RegistryService
 */
public RegistryService getRegistryService() {
    return this.registryService;
}

/**
 * Wether or not the temporary query files should be deletea upon exit
 */
public boolean getCleanTempResources() {
    return this.deleteTempFiles;
}

/**
 * Wether or not the internal service registry should be saved to file upon registration
 */
public boolean getSaveOnRegister() {
    return this.saveOnRegister;
}

/**
 * Wether or not the internal service registry should be loaded from file on startup
 */
public boolean getLoadOnStartup() {
    return this.loadOnStartup;
}

// TODO: This can perhaps be moved outside of the class?

/**
 * Transform query from the global object format to native OWLS-MX Query
 */
public File transformQuery(ServiceQuery query) throws MatchmakerException {
    try {
        File tempFile = File.createTempFile("owlsmx-input-", ".xml");

        // Generic OWLS-MX Query header
        String str = "<?xml version="1.0" encoding="WINDOWS-1252"?>
<owl:RDF xmlns:owl="#http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="#http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owlsmx-service="#http://www.daml.org/services/owl-s/1.1/Service.owl#"
    xmlns:process="#http://www.daml.org/services/owl-s/1.1/Process.owl#"
    xmlns:profile="#http://www.daml.org/services/owl-s/1.1/Profile.owl#"
    xmlns:grounding="#http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
    xmlns:base="#http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
    <owl:Ontology rdf:about="#"/>

        // Add ontology resources (imports)
        str += "<owl:imports rdf:resource="#http://www.daml.org/services/owl-s/1.1/Process.owl#"/>
        str += "<owl:imports rdf:resource="#http://www.daml.org/services/owl-s/1.1/Profile.owl#"/>
        str += "<owl:imports rdf:resource="#http://www.daml.org/services/owl-s/1.1/Grounding.owl#"/>
```
```
for (URI ontology : query.getImports()) {
    str += "<owl:imports rdf:resource=""+ ontology + "" />
"; 
}
str += "</owl:Ontology>"
;
// Generic service description
str += "<service:Service rdf:ID="TARGET_SERVICE"/>
" +
"<service:present rdf:resource="#TARGET_PROFILE"/>
" +
"<service:describes rdf:resource="#TARGET_PROCESS_MODEL"/>
" +
"<service:supports rdf:resource="#TARGET_GROUNDING"/>
" +
"</service:Service>"
;
// Create profile description
int numInputs = query.getInputs().size();
String inputAndOutputText = "";
for (int i = 1; i <= numInputs; i++) {
    inputAndOutputText += "<profile:hasInput rdf:resource="#_INPUT" + i + ">" •
"</profile:Profile>"
;
}
int numOutputs = query.getOutputs().size();
for (int i = 1; i <= numOutputs; i++) {
    inputAndOutputText += "<profile:hasOutput rdf:resource="#_OUTPUT" + i + ">" •
"</profile:Profile>"
;
}
str += inputAndOutputText;
str += "</profile:Profile>"
;
// Create process description
str += "<process:ProcessModel rdf:ID="TARGET_PROCESS_MODEL"/>
" +
"<service:describes rdf:resource="#TARGET_SERVICE"/>
" +
"<process:hasProcess rdf:resource="#TARGET_PROCESS"/>
" +
"</process:ProcessModel>"
;
// Create atomic process description
str += "<process:AtomicProcess rdf:ID="TARGET_PROCESS"/>
" + inputAndOutputText +
"</process:AtomicProcess>"
;
// Define inputs
int i = 0;
for (URI concept : query.getInputs().keySet()) {
    str += "<process:Input rdf:ID="#_INPUT" + (i++) + ">" •
"<process:parameterType rdf:datatype="#http://www.w3.org/2001/XMLSchema#anyURI="#" +
concept.toString() +
"</process:parameterType>"
;
}
// Define outputs
i = 0;
for (URI concept : query.getOutputs()) {
    str += "<process:Output rdf:ID="#_OUTPUT" + (i++) + ">" •
"<process:parameterType rdf:datatype="#http://www.w3.org/2001/XMLSchema#anyURI="#" +
concept.toString() +
"</process:parameterType>"
;
}
// Define WSDL grounding
str += "<grounding:WsdGrounding rdf:ID="#TARGET_GROUNDING"/>
" +
"<service:supportedBy rdf:resource="#TARGET_SERVICE"/>
" +
"</grounding:WsdGrounding>"
;
```java
public @SuppressWarnings("unchecked")
/**
* Transforms the query to the native format and matches services
* @param query The client's query
* @return Vector containing matched services
* @throws MatchmakerException On transformation or matchmaking error
*/
public Vector<Service> performMatch(ServiceQuery query) throws MatchmakerException {
    File tempFile = this.transformQuery(query);
    return this.performMatch(tempFile);
}

/**
* Initializes the matcher, and loads services from file (if option is enabled)
*/
private void initializeMatchmaker() {
    if (this.loadOnStartup && !this.isStateLoaded) {
        this.matchmaker.load();
        this.isStateLoaded = true;
    }
}

/**
* Matches services to query using the OWLS-MX matcher
* @param file File containing the transformed query
* @return Vector containing matched services
* @throws MatchmakerException On transformation or matchmaking error
*/
@SuppressWarnings("unchecked")
public Vector<Service> performMatch(File file) throws MatchmakerException {
    this.initializeMatchmaker();
    Vector<Service> servicesDiscovered = new Vector<Service>();
    try {
        Set<MatchingResult> matches = this.matchmaker.matchRequest(file.toURI());
        if (this.deleteTempFiles) {
            file.delete();
        }
        for (MatchingResult result : matches) {
            // Lookup URI in registry
            Service s = this.registryService.findServiceByUId(result.getServiceId());
            s.setRanking(result.getDegreeOfMatch());
            servicesDiscovered.add(s);
        }
    } catch (Exception e) {
        throw new MatchmakerException("Exception when matching services", e);
    }
    return servicesDiscovered;
```

---

Page 4
/**
 * Add a service to the matchmaker.
 * Note! Don't use this if you don't have to.. It can take a loooong time to add many services!
 * @param service Service to add
 */

public void registerService(Service service) {
    this.initializeMatchmaker();
    this.matchmaker.addService(service.getId(), service.getLocation());
    if (this.saveOnRegister) {
        this.matchmaker.save();
    }
}
package no.ntnu.idi.wisemod.sws4bpel.matchmaking;
import java.io.*;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.*;
import org.mindswap.owl.OWLConfig;
import org.mindswap.owl.OWLFactory;
import org.mindswap.owl.OWLKnowledgeBase;
import org.mindswap.owl.OWLSMXSimilarityMatchmakerImpl;
import org.mindswap.pellet.TuBox.NotUnfoldableException;
import org.apache.log4j.Logger;
import owlsmx.Indexer.SimpleIndex;
import owlsmx.data.ConceptServiceRegistry;
import owlsmx.data.DOM;
import owlsmx.data.ExtendedServiceInformation;
import owlsmx.data.InputServiceContainer;
import owlsmx.data.LocalOntologyContainer;
import owlsmx.data.MatchingResult;
import owlsmx.data.OutputServiceContainer;
import owlsmx.data.ServiceInformation;
import owlsmx.exceptions.ConceptNotFoundException;
import owlsmx.exceptions.MatchingException;
import owlsmx.reasoning.PelletReasoner;
import owlsmx.similaritymeasures.SimilarityMeasure;
import owlsmx.utils.MatchmakerUtils;
import owlsmx.SimilarityMatchmaker;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.rdf.model.RDFNode;
import impl.jena.OWLKnowledgeBaseImpl;

/**
 * Original code taken from the OWLS-MX package
 */
public class OWLSMXSimilarityMatchmakerImpl extends SimilarityMatchmaker {
private static final transient Logger log
= Logger.getLogger(no.ntnu.idi.wisemod.sws4bpel.matchmaking.OWLSMXSimilarityMatchmakerImpl.class);
private org.mindswap.owl.OWLKnowledgeBase base;
private LocalOntologyContainer localOntologyContainer;
private ConceptServiceRegistry registry;
private PelletReasoner reasoner;
private String registryPath = "data\owlsmx\OWLSMX_Retry.dat";
private String ontologyPath = "localOntology.owl";
private Vector new Vector<no.ntnu.idi.wisemod.sws4bpel.domain.Service> serializeList;

private void init(SimilarityMeasure similar) {
this.reason = new PelletReasoner();
this.registry= ConceptServiceRegistry.instanceOf();
this.sim = similar;
this.serializeList = new Vector<no.ntnu.idi.wisemod.sws4bpel.domain.Service>();
this.retryCounter = 0;
this.allowedRetries = 2;
// TODO: These lines (really only OWLFactory.createKB(), but it is also called from within LocalOntologyContainer
// crash OWLS VM!!!!
this.base = OWLFactory.createKB();//= new OWLKnowledgeBaseImpl();
this.localOntologyContainer = new LocalOntologyContainer();
}

/**
 * Initializes the matcher without any similarity measure and
 * hence purely semenatic matching
 */
public OWLSMXSimilarityMatchmakerImpl() {
init(null);
}
/**
 * Initializes the matchmaker with the given similarity measure
 * @param similar Similarity measure to be used
 */
 public OWLSMXSimilarityMatchmakerImpl(SimilarityMeasure similar)
  { init(similar); }

 /*
 * Initializes the matchmaker with the given similarity measure
 * @param sim Similarity measure to be used
 */
 public OWLSMXSimilarityMatchmakerImpl(short sim) {
  SimilarityMeasure similar = switchSimilarityMeasure(sim);
  init(similar); }

 /*********************************************************************************/

 /** Number of retry attempts on connection failures */
 private int allowedRetries;

 /** Internal retry counter */
 private int retryCounter;

 private int tmpSimMeasure;

 public void setTmpSimMeasure(int value) {
  this.tmpSimMeasure = value;
  super.setSimilarityMeasure((short)value);
 }

 public int getTmpSimMeasure() {
  return (int)super.getSimilarityMeasureType();
 }

 /**
 * Set the number of retries allowed on connection failure
 * @param allowedRetries Number of retry attempts on connection failure
 */
 public void setAllowedRetries(int allowedRetries) {
  this.allowedRetries = allowedRetries;
 }

 /**
 * Get the number of retries allowed on connection failure
 * @return Number of retry attempts on connection failure
 */
 public int getAllowedRetries() {
  return allowedRetries;
 }

 /**
 * Checks if a connection retry is allowed.
 * @li <code>true</code> the retry counter is incremented.
 * @li <code>true</code> if a retry is allowed, else <code>false</code>
 */
 protected boolean retryAllowed() {
  if (this.retryCounter < this.allowedRetries) {
    this.retryCounter++;
    return true;
  }
  return false;
 }

 public void setRegistryPath(String registryPath) {
  this.registryPath = registryPath;
 }

 public void setOntologyPath(String ontologyPath) {
  this.ontologyPath = ontologyPath;
 }

protected void setBase(OWLKnowledgeBase base) {
    this.base = base;
}

protected void setLocalOntologyContainer(LocalOntologyContainer localOntologyContainer) {
    this.localOntologyContainer = localOntologyContainer;
}

protected void setRegistry(ConceptServiceRegistry registry) {
    this.registry = registry;
}

protected void setReason(PelletReasoner reason) {
    this.reason = reason;
}

protected void setSerializeList(Vector<no.ntnu.idi.wisemod.sws4bpel.domain.Service> serializeList) {
    this.serializeList = serializeList;
}

public String getOntologyPath() {
    return ontologyPath;
}

public String getRegistryPath() {
    return registryPath;
}

protected OWLKnowledgeBase getBase() {
    return base;
}

protected LocalOntologyContainer getLocalOntologyContainer() {
    return localOntologyContainer;
}

protected ConceptServiceRegistry getRegistry() {
    return registry;
}

protected PelletReasoner getReason() {
    return reason;
}

protected Vector<no.ntnu.idi.wisemod.sws4bpel.domain.Service> getSerializeList() {
    return serializeList;
}

/**
 * Save serializableList to file
*/
public void saveToFile() {
    ObjectOutput out = null;
    try {
        out = new ObjectOutputStream(new FileOutputStream(this.registryPath));
        out.writeObject(this.serializeList);
        } catch (Exception e) {
            log.fatal("Could not save ServiceRegistry to file " + this.registryPath,e);
        } finally {
            try {
                if(out != null) out.close();
                } catch (Exception e) {
        }
    }

    /**
    * Load serializableList from file
    */
    * TODO: This is not very scalable!!!!
    */
public void loadFromFile() {
    ObjectInputStream in = null;
    this.serializeList = new Vector<no.ntnu.idi.wisemod.sws4bpel.domain.Service>();
    try {
    } finally {
        } catch (Exception e) {
    }
File = F:\Backup\Documents\Skole\master\sws4bpel\src\main\java\no\ntnu\idi\wisemod\sws4bpel\matchmaking\OWLSMXSimilar

    @SuppressWarnings("unchecked")
    public void addService(Integer id, URI uri) {
        try {
            org.mindswap.owl.OWLOntology ontology = null;
            try {
                ontology = base.read(uri);
            }
            catch (FileNotFoundException e) {
                if (in != null) in.close();
                catch (Exception e) {}
            }
            Service service = ontology.getService();
            Vector<URI> inputs = MatchmakerUtils.getInputStream(service.getProfile().getInputs());
            Vector<URI> outputs = MatchmakerUtils.getOutputStream(service.getProfile().getOutputs());
            no.ntnu.idi.wisemod.sws4bpel.domain.Service newService
                = new no.ntnu.idi.wisemod.sws4bpel.domain.Service(uri, id, inputs, outputs);
            this.serializeList.add(newService);
        }
        catch (Exception e) {
            log.error("Could not add service from this URI " +uri.toString(),e);
            finally {
                this.base.unload(uri);
            }
        }
    }

    public String toString() {
        return registry.toString();
    }

    public void print() {
        System.err.println(this.getClass().toString());
    }

/** Retrieves Services in the registry that are registered at certain concepts */

/* @param isInput if it's an input concept */
/* @param classes Set of OWLClass(es) with the concepts to be retrieved */
/* @param concept concept in the query we want candidate matches for */
/* @return SortedSet of ExtendedServiceInformation(s) with the candidate services for these concepts */

@throws URISyntaxException
@throws ConceptNotFoundException
*/

protected SortedSet getServicesFromSet(boolean isInput, Set classes, OntClass concept, int degreeOfMatch) throws URISyntaxException, ConceptNotFoundException {
    this.save();
    Set result = new TreeSet();
    Iterator iter = classes.iterator();
    Set services;
    ServiceInformation info;
    ServiceIterator serviceIterator;
    RDFNode clazz;
    while (iter.hasNext()) {
        clazz = (RDFNode) iter.next();
        services = registry.getServices(isInput, clazz.toString());
        serviceIterator = services.iterator();
        while (serviceIterator.hasNext()) {
            info = (ServiceInformation) serviceIterator.next();
            try {
                if (useSyntacticFilter())
                    result.add(new ExtendedServiceInformation(info, degreeOfMatch, reason.unfoldTerm(clazz)));
                else {
                    //purely semantic matching doesn't need the unfolded concept
                    result.add(new ExtendedServiceInformation(info, degreeOfMatch, ""));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    return result;
}

/** Computes the set of candidates for a given service output concept */

/* @param concept URI (String) of the given concept */
/* @return SortedSet of ExtendedServiceInformation items */

@throws URISyntaxException If the URI is invalid
@throws ConceptNotFoundException If the concept is not found
*/

protected SortedSet getServicesForOutputConcept(String concept) throws URISyntaxException, ConceptNotFoundException {
    OntClass clazz = localOntologyContainer.getOntClass(concept);
    Set equivalent = reason.retrieveEquivalentClasses(clazz);
    Set parents = reason.retrieveDirectParentClasses(clazz, equivalent);
    Set ancestors = reason.retrieveAncestorClasses(clazz, equivalent, parents);
    Set children = reason.retrieveDirectSubClasses(clazz, equivalent);
    Set descendants = reason.retrieveDescendantClasses(clazz, equivalent, children);

    SortedSet candidates = new TreeSet();
    candidates.addAll(getServicesFromSet(false, equivalent, clazz, SimilarityMatchmaker.EXACT));
    candidates.addAll(getServicesFromSet(false, children, clazz, SimilarityMatchmaker.PLUGIN));
    candidates.addAll(getServicesFromSet(false, descendants, clazz, SimilarityMatchmaker.SUBSUMES));
    candidates.addAll(getServicesFromSet(false, parents, clazz, SimilarityMatchmaker.SUPERSUMED_BY));
    candidates.addAll(getServicesFromSet(false, reason.retrieveRemainingClasses(clazz, equivalent, parents, ancestors, children, descendants), clazz, SimilarityMatchmaker.NEAREST_NEIGHBOUR));

    return candidates;
/** Computes the set of candidates for a given service input concept */

protected SortedSet getServicesForInputConcept(String concept) throws URISyntaxException, ConceptNotFoundException {
    OntClass clazz = localOntologyContainer.getClass(concept);
    Set equivalent = reason.retrieveEquivalentClasses(clazz);
    Set ancestors = reason.retrieveAllAncestorClasses(clazz);

    SortedSet candidates = new TreeSet();
    candidates.addAll(getServicesFromSet(true, equivalent, clazz, SimilarityMatchmaker.EXACT));
    candidates.addAll(getServicesFromSet(true, ancestors, clazz, SimilarityMatchmaker.PLUGIN));
    return candidates;
}

/** Computes the set of candidates for multiple service output concepts */

protected Map getOutputCandidates(Vector<OntClass> outputConcepts) throws URISyntaxException, MatchingException {
    if (outputConcepts.size() <= 0) {
        return registry.getAllServices(false);
    }
    OutputServiceContainer resultContainer = new OutputServiceContainer();
    resultContainer.addServices(getServicesForOutputConcept(String.valueOf(outputConcepts.get(0)).toString()));
    return resultContainer.getServicesMap();
}

@SuppressWarnings("unchecked")
protected Map addEmptyInputs(Map<OntClass, Map<URI, List<ExtendedServiceInformation>>> map) {
    Map<URI, List<ExtendedServiceInformation>> newMap = new HashMap();
    newMap.put(map.get(inputConcept).get(0), new ArrayList());
    return newMap;
}

@SuppressWarnings("unchecked")
protected Map get心思Candidates(Vector<OntClass> inputConcepts) throws URISyntaxException, ConceptNotFoundException {
    if (inputConcepts.size() <= 0) {
        return registry.getAllServicesWithoutInput();
    }
    InputServiceContainer inputs = new InputServiceContainer();
    SortedSet<OntClass> conceptSet = new TreeSet();
    for (OntClass concept : inputConcepts) {
        inputs.addServices(concept);
    }
    return inputs.getServicesForInputConcept(String.valueOf(inputConcepts.get(i)).toString());
}
```java
protected Set getServiceInformationFromDOM(Set serviceDOMs) {
    Set result = new HashSet();
    Iterator iter = serviceDOMs.iterator();
    while (iter.hasNext()) {
        result.add((DOM)iter.next()).getBestDegree();
    }
    return result;
}

protected SortedSet inputCandidatesToResult(Map inputCandidates) {
    SortedSet result = new TreeSet();
    ExtendedServiceInformation inInfo;
    Map.Entry me;
    Iterator iter = inputCandidates.entrySet().iterator();
    while (iter.hasNext()) {
        me = (Map.Entry)iter.next();
        inInfo = (ExtendedServiceInformation)me.getValue();
        result.add(new MatchingResult(inInfo, inInfo.unfoldedconcept, ""));
    }
    return result;
}

protected SortedSet semanticMatch(Vector inConcepts, Vector outConcepts) throws URISyntaxException, MatchingException {
    Map input = getInPutCandidates(inConcepts);
    if ((outConcepts == null) || (outConcepts.size()<=0)) {
        return inputCandidatesToResult(input);
    }
    Map output = getOutPutCandidates(outConcepts);
    SortedSet result = new TreeSet();
    Map.Entry me;
    Integer ID;

    ExtendedServiceInformation inInfo, outInfo;

    DOM degree;
    Iterator iter = input.entrySet().iterator();
    while (iter.hasNext()) {
        me = (Map.Entry)iter.next();
        ID = (Integer)me.getKey();
        if (output.containsKey(ID)) {
            inInfo = (ExtendedServiceInformation)me.getValue();
            degree = ((DOM) output.get(ID));
        }
    }
```

/**
 * Syntactic Filter that adjusts the semantic degree of match according to the actual similarity
 */

@suppressWarnings("unchecked")

protected SortedSet syntacticFilters(Service queryService, Vector queryInputs, Vector queryOutputs, SortedSet semanticResults)

throws NotUnfoldableException, URISyntaxException, MatchingException {

SortedSet result = new TreeSet();

String input = reason.unfoldURIs(queryInputs);
String output = reason.unfoldURIs(queryOutputs);

double sim_input, sim_output;

MatchingResult info;

while (iter.hasNext()) {
    sim_input = 0.0;
    sim_output = 0.0;
    info = (MatchingResult) iter.next();

    if (!info.unfoldedInput.equals(""))
        sim_input = sim.computeSimilarity(queryService.getURI().toString(), input,""+info.serviceID, info.unfoldedInput.size())
    if (queryOutputs.size()>0)
        sim_output = sim.computeSimilarity(queryService.getURI().toString(), output,""+info.serviceID, info.unfoldedOutput);

    info.similarity = (sim_input + sim_output) / 2;

    if (info.similarity < getSyntacticThreshold()) {
        if (! (info.degreeOfMatch instanceof SimilarityMatchmaker.NEAREST_NEIGHBOUR) ||
            (info.degreeOfMatch == SimilarityMatchmaker.SUBSUMED_BY))
            info.degreeOfMatch = SimilarityMatchmaker.FAIL;
        else
            result.add(info);
    } else
        result.add(info);
}

return result;

/*
 * Updates the reasoner with the local ontology
 */

protected void updateReasoner() {

    reason.clear();
    reason.load((OntModel)localOntologyContainer.getOntology().getImplementation());
}

/*
 * Little helper function which performs the actual adding of a service
 */

@suppressWarnings("unchecked")

protected void addService(Service service, Vector inputs, Vector outputs) {

    if (inputs.size() == 0)
        return;

    double sim_input, sim_output;

    // Compute similarity

    sim_input = sim.computeSimilarity(service.getURI().toString(), inputs.size());
    sim_output = sim.computeSimilarity(service.getURI().toString(), outputs.size());

    // Update reasoner

    reason.addService(service, sim_input, sim_output);
}

/
protected void addService(Integer serviceID, org.mindswap.owls.Ontology onto, Vector inputurilist, Vector outputurilist) {
    try {
        Set conceptsToAdd = new HashSet();
        conceptsToAdd.addAll(inputurilist);
        conceptsToAdd.addAll(outputurilist);
        localOntologyContainer.processClasses(base, conceptsToAdd);
        updateReasoner();
    } catch (Exception e) { e.printStackTrace(); }
}

/* (non-Javadoc)*/
public SortedSet matchRequest(URI profileURI) throws MatchingException {
    try {
        org.mindswap.owls.Ontology onto = base.read(profileURI);
        Service service = onto.getService();
        Vector inputurilist=MatchmakerUtils.getURIList(service.getProfile().getInputs());
        Vector outputurilist=MatchmakerUtils.getURIList(service.getProfile().getOutputs());
        Set conceptsToAdd = new HashSet();
        conceptsToAdd.addAll(inputurilist);
        conceptsToAdd.addAll(outputurilist);
        localOntologyContainer.processClasses(base, conceptsToAdd);
        updateReasoner();
        base.unload(profileURI);
    } catch (MatchingException e) {
        throw new MatchingException(e.toString());
    }
}

/* (non-Javadoc)*/
public void removeService(Integer integer) {
    registry.removeService(integer);}

/* (non-Javadoc)*/
public boolean load() {
    boolean result = LocalOntologyContainer.load();
    this.loadFromFile();
    return result;
}

/* (non-Javadoc)*/
public boolean save() {
    boolean result = localOntologyContainer.save();
    this.saveToFile();
    return result;
}
public void clear() {
    File file = new File(this.registryPath);
    if (file.exists()) {
        file.delete();
    }
    file = new File(this.ontologyPath);
    if (file.exists()) {
        file.delete();
    }
    ConceptServiceRegistry.instanceOf().clear();
    SimpleIndex.instanceOf().clear();
}

/**
 * Enables the support for profile hierarchies
 *
 * @param speedyButProblematic disables general check if a profile is valid - hence profile hierarchies don't cause problems but might lead to secondary problems
 */
public void enableProfileHierarchies(boolean speedyButProblematic) {
    if (speedyButProblematic)
        OWLConfig.setStrictConversion(false);
    else
        base.setReasoner("RDFS");
}

/**
 * Disables the support for profile hierarchies
 */
public void disableProfileHierarchies() {
    base.setReasoner(null);
    OWLConfig.setStrictConversion(true);
}

```java
package no.ntnu.idi.wisemod.sws4bpel.selection;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;
import java.util.Vector;
import java.util.Collections;

/**
 * Generic service selection, based on the ranking from the matchmaker
 */
public class GenericSelectionService implements SelectionService {
    // TODO: Implement QoS evaluation, the algorithm must set the ranking property of the Service, smaller value is better
    // TODO: The Query must also be altered here... the inputs must be bound to the input concepts of the service...
    // initially they are bound to the ontology concepts...
    /**
     * Rank services by their QoS properties
     * @param query ServiceQuery that specifies the requirements of the client
     * @param services Service candidates
     * @return Sorted vector containing the highest ranking services first (uses the Service.ranking property)
     */
    public Vector<Service> rankServices(ServiceQuery query, Vector<Service> services) {
        Collections.sort(services);
        return services;
    }

    /**
     * Select a service by QoS properties
     * @param query ServiceQuery that specifies the requirements of the client
     * @param services Service candidates
     * @return Selected service
     */
    public Service selectService(ServiceQuery query, Vector<Service> services) {
        services = this.rankServices(query, services);
        return services.get(0);
    }
}
```
package no.ntnu.idi.wisemod.sws4bpel.selection;

/**
 * Generic exception class for the invocation package
 */
public class SelectionException extends Exception {
    public SelectionException() {
    }

    public SelectionException(String string) {
        super(string);
    }

    public SelectionException(String string, Throwable throwable) {
        super(string, throwable);
    }

    public SelectionException(Throwable throwable) {
        super(throwable);
    }
}
package no.ntnu.idi.wisemod.sws4bpel.selection;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;

import java.util.Vector;

/**
 * This service provides the functionality for selecting and ranking service candidates by
 * the Quality of Service attributes provided by the client through the ServiceQuery
 */
public interface SelectionService {

    /**
     * Select a service by QoS properties
     *
     * @param query ServiceQuery that specifies the requirements of the client
     * @param services Service candidates
     * @return Selected service
     */
    public Service selectService(ServiceQuery query, Vector<Service> services);

    /**
     * Rank services by their QoS properties
     *
     * @param query The Client's query
     * @param services Service candidates
     * @return Sorted vector containing the highest ranking services first (uses the Service.ranking property)
     */
    public Vector<Service> rankServices(ServiceQuery query, Vector<Service> services);
}
package no.ntnu.idi.wisemod.sws4bpel.invocation;

/**
 * General exception class for the invocation package
 */

public class InvocationException extends Exception {
    public InvocationException() {
    }
    public InvocationException(String string) {
        super(string);
    }
    public InvocationException(String string, Throwable throwable) {
        super(string, throwable);
    }
    public InvocationException(Throwable throwable) {
        super(throwable);
    }
}
package no.ntnu.idi.wisemod.sws4bpel.invocation;

import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceResult;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;

/**
 * Interface that defines an Invocation Service. That is a service that executes other services returning a ServiceResult instance containing the results.
 * This interface should be implemented for each technology / service language
 */

public interface InvocationService {

    /**
     * Invokes the provided service and returns an instance of ServiceResult
     *
     * @param query The clients query, containing the input values
     * @param service The service that is to be executed
     * @return ServiceResult instance containing the results
     * @throws InvocationException if the execution fails
     *
     */
    public ServiceResult invoke(ServiceQuery query, Service service) throws InvocationException;
}


```java
package no.ntnu.idi.wisemod.swt4bpel.invocation;

import no.ntnu.idi.wisemod.swt4bpel.domain.ServiceResult;
import no.ntnu.idi.wisemod.swt4bpel.domain.Service;
import no.ntnu.idi.wisemod.swt4bpel.domain.ServiceQuery;
import java.net.URI;
import java.util.HashMap;
import java.io.FileNotFoundException;
import org.apache.log4j.Logger;

import org.mindswap.owls.OWLSSFactory;
import org.mindswap.owls.process.execution.ProcessExecutionEngine;
import org.mindswap.owls.process.InputList;
import org.mindswap.owls.process.Process;
import org.mindswap.owls.process.OutputList;
import org.mindswap.owl.ServiceQuery;
import org.mindswap.query.ValueMap;

public class OWLSInvocationService implements InvocationService {
    /* Log4J logger */
    private static final transient Logger log
        = Logger.getLogger(owlswisemod.swt4bpel.invocation.OWLSInvocationService.class);

    public OWLSInvocationService() {
        this.allowedRetries = 2;
        this.retryCounter = 0;
    }

    public void setAllowedRetries(int allowedRetries) {
        this.allowedRetries = allowedRetries;
    }

    public int getAllowedRetries() {
        return allowedRetries;
    }

    protected boolean retryAllowed() {
        if (this.retryCounter < this.allowedRetries) {
            this.retryCounter++;
            return true;
        }
        return false;
    }

    /* Specified by {invoke no.ntnu.idi.wisemod.swt4bpel.invocation.InvocationService#invoke}
     */
```
    "@param query The query from the client, containing input values
    "@return Result of the service invocation or <code>null</code> if there are no results
    "@throws InvocationException If invocation fails
    
    public ServiceResult invoke(ServiceQuery query, Service service) throws InvocationException {
        log.debug("------> READY TO INVOKED ------");
        if (service == null) {
            throw new InvocationException("Invalid service, service was null");
        }
        if (query == null) {
            throw new InvocationException("Invalid query, query was null");
        }
        ServiceResult result = new ServiceResult();
        service = query.getOperation(). getService();
        if (service == null) {
            throw new InvocationException("Invalid service, service was null");
        }
        log.debug("------> SERVICE describe ------");
        org.mindswap.owl.Service svc = service.getOperation().getService();
        svc = query.getOperation().getService();
        if (svc == null) {
            throw new InvocationException("Invalid service, service was null");
        }
        log.debug("------> HELLO ------");
        try {
            ProcessExecutionEngine execEngine = OWLSFactory.createExecutionEngine();
            OWLKnowledgeBase kb = OWLSFactory.createKB();
            log.debug("------>READY TO PARSE SERVICE ------");
            while (this.retryAllowed()) {
                try {
                    parsedService = kb.readService(svc.getLocation().toURI());
                } catch (FileNotFoundException e) {
                    if (this.retryAllowed()) {
                        this.retryCounter = 0;
                        throw new InvocationException("Could not access service description", e);
                    }
                }
            }
            org.mindswap.owlProcess Process parsedProcess = parsedService.getService();
            ValueMap inputValueMap = new ValueMap();
            log.debug("------> READY TO GET INPUTS ------");
            for (Object obj : inputList) {
                Input input = (Input)obj;
                // TODO: Need to map query inputs to service inputs in a better way....
                // Now there might be a difference between the ontology concepts used in the query, and the
                // inputs in the service... matching against URI is NOT enough!!!!!!!
                // Tried to get proper type from input.. but all I got was the default #Input type from the OWL-S ontology &/&
                if (queryInputMap.containsKey(input.getURI())) {
                    log.debug("""ADDDED INPUT """" + input.getURI() + """" + queryInputMap.get(input.getURI()));
                    inputValueMap.setDataValue(input, queryInputMap.get(input.getURI()));
                }
            }
            log.debug("------> Executing service """" + service.getLocation());
            log.debug("------> Parsed process: """" + parsedProcess.toType());
            log.debug("------> Number of inputs: """" + inputValueMap.size());
            ValueMap outputValueMap = execEngine.execute(parsedProcess, inputValueMap);
            OutputList outputList = parsedProcess.getOutputs();
            if (outputList.size() != 0) {
                OWLValue outputValue = outputValueMap.get(output);
                if (outputValue.isDataValue()) {
                    OWLDataValue data = outputValueMap.getDataValue(output);
                    log.debug("--------- FOUND OWLDataValueOUTPUT: " """ + individual.toString(RDF));
                    result.addDataValue(data, outputValueMap.getDataValue(output));
                } else {
                    OWLIndividual individual = outputValueMap.getIndividualValue(output);
                    log.debug("--------- Found OWLIndividual OUTPUT: " + individual.toString(RDF));
                }
            }
        }
        return result;
    }
}
```java
result.addResult(output.getURI(), individual.toRDF());

return result;

return null;

} catch (Exception e) {
    throw new InvocationException("Exception when executing " + service.getLocation(), e);
}
```
package no.ntnu.idi.wisemod.sws4bpel.invocation;

import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceResult;
import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;

import edu.cmu.atlas.osvm.OWLSVMApplication;
import edu.cmu.atlas.osvm.PerformContext;
import edu.cmu.atlas.osvm.OWLSVM;
import edu.cmu.atlas.osvm.impl.PerformContextImpl;

import edu.cmu.atlas.Axis1_1.service.ServiceGrounding;
import edu.cmu.atlas.Axis1_1.service.ServiceGroundingList;
import edu.cmu.atlas.Axis1_1.service.OWLSServiceModel;
import edu.cmu.atlas.Axis1_1.service.ServiceList;
import edu.cmu.atlas.Axis1_1.process.Process;
import edu.cmu.atlas.Axis1_1.process.Perform;
import edu.cmu.atlas.Axis1_1.process.AtomicProcess;
import edu.cmu.atlas.Axis1_1.process.InputList;
import edu.cmu.atlas.Axis1_1.process.Input;
import edu.cmu.atlas.Axis1_1.process.OutputList;
import edu.cmu.atlas.Axis1_1.process.Output;
import edu.cmu.atlas.Axis1_1.parser.OWLSVParse;

import java.net.URI;
import java.net.URIException;
import java.util.HashMap;

import org.apache.log4j.Logger;

/**
 * Implementation of InvocationService for OWL-S and OWLS VM.
 */
public class OWLSVMInvocationService implements OWLSVMApplication, InvocationService {

    public static final transient Logger log
        = Logger.getLogger(no.ntnu.idi.wisemod.sws4bpel.invocation.OWLSVMInvocationService.class);

    /** OWLSVM Implementation instance */
    private OWLSVM owlsvm;

    /** Shared client query, for use with the callback methods */
    private ServiceQuery currentQuery;

    /** Shared service result, for use within the callback methods */
    private ServiceResult currentResult;

    /** Constructor */
    @param owlsvm OWLSVM implementation instance

    public OWLSVMInvocationService(OWLSVM owlsvm) {
        this.owlsvm = owlsvm;
        this.owlsvm.setOsvmApplication(this);
        this.setOsvmApplication(owlsvm);
    }

    /**
     * Set the OWLS VM implementation instance
     */
    @param owlsvm OWLSVM Instance

    public void setOsvm(OWLSVM owlsvm) {
        this.owlsvm = owlsvm;
        this.owlsvm.setOsvmApplication(this);
    }

    /**
public synchronized ServiceResult invoke(ServiceQuery query, ServiceService) throws InvocationException {
    OWLSServiceParser parser = new OWLSServiceParser();
    OWLSServiceModel serviceModel = null;
    log.debug("**OWLSServiceModel**");
    log.debug("**OWLSServiceParser**");
    try {
        ServiceList serviceList = serviceModel.getServiceList();
        ServiceOVM serviceOVM = serviceList.getService(0);
        try {
            this.owlsvm.executeService(serviceOVM);
            return this.getCurrentResult();
        }
    }
    catch (Exception e) {
        throw new InvocationException("Exception when parsing model for service: " + service.getLocation().toString());
    }
    log.debug("**Done parsing service model <---**");
}
```java
public OWLSVMInvocation selectGrounding(ServiceGroundingList serviceGroundingList) {
    log.debug("-----SELECTING GROUNDING ------> " + serviceGroundingList.getNthServiceGrounding(0).getURI());
    // TODO: Should be improved... or there should only be _one_ grounding in the list
    return serviceGroundingList.getNthServiceGrounding(0);
}
```

```java
public Process selectProcess(ProcessList processList) {
    log.debug("-----SELECTING PROCESS ------> " + processList.getNthProcess(0).getURI());
    // TODO: Should be improved... or there should only be _one_ process in the list
    return processList.getNthProcess(0);
}
```

```java
public PerformContext setInputs(Perform perform) {
    log.debug("---------- INSIDE SETINPUTS ----------");
    PerformContext context = new PerformContextImpl();
    AtomicProcess atomic = (AtomicProcess) perform.getProcess();
    HashMap<String, String> queryInputMap = this.getQueryInputMap();
    // TODO: This requires an exact match between the URIs, no room for deduction?
    InputList inputList = atomic.getInputList();
    for (int i = 0; i < inputList.size(); i++) {
        Input input = inputList.getNthInput(i);
        try {
            URI inputURI = new URI(input.getInputURI());
            if (queryInputMap.containsKey(inputURI)) {
                context.setInputValue(input, queryInputMap.get(inputURI));
            }
        } catch (URISyntaxException e) {
            log.error("Invalid URI. " + inputURI());

        }
    }
    return context;
}
```

```java
* Process the outputs of the process that was executed. The values of the outputs are accessed
  * using the PerformContext object.
```
* Note! This is an OWLS VM Callback method

* @param perform The process and actions that have been performed by the service
* @param context The PerformContext containing inputs

public void processOutputs(Perform perform, PerformContext context) {
    this.setCurrentResult(null);
    log.debug("-------- INSIDE processOutputs ---------------");
    ServiceResult result = new ServiceResult();
    AtomicProcess atomic = (AtomicProcess) perform.getProcess();
    OutputList outputList = atomic.getOutputList();
    for (int i = 0; i < outputList.size(); i++) {
        Output output = outputList.getNthOutput(i);
        try {
            result.addResult(new URI(output.getURI()), context.getOutputValue(output));
        } catch (URISyntaxException e) {
            log.error("Invalid URI: " + output.getURI());
        }
    }
    this.setCurrentResult(result);
}
File - F:\Backup\Documents\Skole\master\sws4bpel\src\main\resources\spring.xml

<?xml version="1.0" encoding="UTF-8"?>
<beans>

  <!-- User editable properties -->
  <bean id="userProperties" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location">sws4bpel.properties</property>
  </bean>

  <!-- Hibernate Session Factory -->
  <bean id="sessionSessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="hibernateProperties" ref="hibernateProperties"/>
    <property name="mappingResources">
      <list>
        <value>no/ntnu/idi/wisemod/sws4bpel/domain/Service.hbm.xml</value>
        <value>no/ntnu/idi/wisemod/sws4bpel/domain/Session.hbm.xml</value>
      </list>
    </property>
  </bean>

  <!-- Hibernate Properties -->
  <bean id="hibernateProperties" class="org.springframework.beans.factory.config.PropertiesFactoryBean">
    <property name="properties">
      <props>
        <ref key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</ref>
        <ref key="hibernate.dialect">${hibernate.dialect}</ref>
        <ref key="hibernate.show_sql">${hibernate.show_sql}</ref>
        <ref key="hibernate.c3p0.minPoolSize">5</ref>
        <ref key="hibernate.c3p0.maxPoolSize">20</ref>
        <ref key="hibernate.c3p0.timeout">600</ref>
        <ref key="hibernate.c3p0.testConnectionOnCheckout">false</ref>
        <ref key="hibernate.cache.use_query_cache">false</ref>
        <ref key="hibernate.cache.use_minimal_puts">false</ref>
        <ref key="hibernate.max_fetch_depth">3</ref>
      </props>
    </property>
  </bean>

  <!-- ServiceLocator Bean -->
  <bean id="serviceLocator" class="org.springframework.beans.factory.config.ServiceLocatorFactoryBean">
    <property name="serviceLocatorInterface" value="no.ntnu.idi.wisemod.sws4bpel.core.ServiceLocator"/>
    <property name="serviceLocatorExceptionClass" value="no.ntnu.idi.wisemod.sws4bpel.core.ServiceLocatorException"/>
  </bean>

  <!-- ServiceDAO Bean -->
  <bean id="serviceDAO" ref="sessionSessionFactory"/>
  <bean id="serviceDAOBean" class="no.ntnu.idi.wisemod.sws4bpel.registry.dao.hibernate.ServiceHibernateDAO">
    <property name="sessionFactory" ref="hibernateSessionFactory"/>
  </bean>

  <!-- Registry Service Bean -->
  <bean id="registryBean" class="no.ntnu.idi.wisemod.sws4bpel.registry.RegistryServiceImpl"/>
  <bean id="registryDAO" ref="sessionDAO"/>
</beans>
<bean id="owlsmxMatchmakerBean" class="no.ntnu.idi.wisemod.sws4bpel.matchmaking.OWLSMXSimilarityMatchmakerImpl">
    <property name="tmpSimMeasure"><value>${mm.owlsmx.similarityMeasure}</value></property>
    <property name="registryPath"><value>${mm.owlsmx.registryPath}</value></property>
    <property name="allowedRetries"><value>${mm.owlsmx.allowedRetries}</value></property>
</bean>

<bean id="matchmakerBean" class="no.ntnu.idi.wisemod.sws4bpel.matchmaking.OWLSMXMatchmakerServiceAdapter">
    <property name="cleanTempResources"><value>${mm.cleanTempResources}</value></property>
    <property name="loadOnStartup"><value>${mm.loadOnStartup}</value></property>
    <property name="saveOnRegister"><value>${mm.saveOnRegister}</value></property>
    <property name="registryService" ref="registryBean"/>
    <property name="matchmaker" ref="owlsmxMatchmakerBean"/>
</bean>

<bean id="invocationBean" class="no.ntnu.idi.wisemod.sws4bpel.invocation.OWLSInvocationService">
    <property name="allowedRetries"><value>${invocation.allowedRetries}</value></property>
</bean>

<bean id="selectionBean" class="no.ntnu.idi.wisemod.sws4bpel.selection.GenericSelectionService"/>

</beans>
# Main config file for deployment

### Database Settings

# JDBC Driver class with full package name
db.driverClass=org.hsqldb.jdbcDriver

# Datasource connection URI
db.uri=jdbc:hsqldb:file:data/database/registry

# Datasource username
db.username=sa

# Datasource password
db.password=

### Hibernate Settings

# Hibernate SQL dialect, must match the database used
hibernate.dialect=org.hibernate.dialect.HSQLDialect

# Write SQL queries to Log? (true/false)
hibernate.show_sql=true

# What action to perform when translating Hibernate HBM definitions to SQL DDL (update/create)
# Note! Create overwrites data on each run!
hibernate.hbm2ddl=update

### Matchmaker settings

# Delete temporary matchmaker files? (true/false)
mm.cleanTempResources=true

# Save registry when a new service is added? (true/false)
mm.saveOnRegister=true

# Load service registry on matchmaker startup? (true/false)
mm.loadOnStartup=true

### OWLS-MX Specific

# Path to file-based OWLS-MX registry (relative or full)
mm.owlsmx.registryPath=data/owlsmx/OWLSMX_Registry.dat

# Number of retries before a connection exception is thrown when trying to access a remote service description
mm.owlsmx.allowedRetries=3

# OWLS-MX SimilarityMeasure, see OWLS-MX JavaDoc for full list and description
mm.owlsmx.similarityMeasure=1

### Invocation settings

# Number of retries before a connection exception is thrown when trying to invoke a remote service
invocation.allowedRetries=3
package no.ntnu.idi.wisemod.sws4bpel.ws;

import no.ntnu.idi.wisemod.sws4bpel.core.SWS4BPELService;
import no.ntnu.idi.wisemod.sws4bpel.domain.Service;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceQuery;
import no.ntnu.idi.wisemod.sws4bpel.domain.ServiceResult;
import no.ntnu.idi.wisemod.sws4bpel.domain.Session;
import javax.java.WebService;
import javax.java.WebMethod;
import javax.java.WebParam;
import javax.java.WebResult;
import java.util.Vector;
import java.util.Collection;
import org.apache.log4j.Logger;

public class SWS4BPELWebService {
    // TODO: Make logging compatible with JBoss deployment! Throws a mile-long exception at deploy-time for some reason.
    private static final transient Logger log = Logger.getLogger(no.ntnu.idi.wisemod.sws4bpel.ws.SWS4BPELWebService.class);

    private SWS4BPELService sws4bpel;

    public void setSWS4BPELService(SWS4BPELService sws4bpel) {
        this.sws4bpel = sws4bpel;
    }

    public SWS4BPELService getSWS4BPELService() {
        return sws4bpel;
    }

    @WebMethod(operationName = "RegisterService")
    public Service registerService(Service service) {
        log.debug("-------------- Web Service ready to call registerService");
        logger.infoReceivedData(service.toString());
        Service regged = this.sws4bpel.registerService(service);
        log.debug("-------------- \nReturned data: "+regged.toString());
        return regged;
    }

    @WebMethod(operationName = "RemoveService")
    public void removeService(Integer integer) {
        log.debug("-------------- Web Service ready to call removeService with id "+integer);
        this.sws4bpel.removeService(integer);
    }

    @WebMethod(operationName = "PerformMatch")
    @WebResult(name = "ServiceList")
    public Collection<Service> performMatch(ServiceQuery serviceQuery) {
        log.debug("-------------- Web Service ready to call performMatch");
        log.debug("-------------- \nReceived data: "+serviceQuery.toString());
        return this.sws4bpel.performMatch(serviceQuery);
    }

    @WebMethod(operationName = "PerformAutomaticProcess")
    public ServiceResult performAutomaticProcess(ServiceQuery serviceQuery) {
        log.debug("-------------- Web Service ready to call performAutomaticProcess");
        log.debug("-------------- \nReceived data: "+serviceQuery.toString());
        ServiceResult result = this.sws4bpel.performAutomaticProcess(serviceQuery);
        log.debug("-------------- \nReturned data: "+result.toString());
    }
}
public Integer initiateManualProcess(ServiceQuery serviceQuery) {
    log.debug("----------> Web Service ready to call performManualProcess");
    Integer sessionId = this.sws4bpel.performManualProcess(serviceQuery);
    log.debug("----------> InReceived data: " + sessionId);
    return sessionId;
}

@WebMethod(operationName = "ResumeManualProcess")
public ServiceResult resumeManualProcess(Integer sessionId, Integer selectedServiceId) {
    log.debug("----------> Web Service ready to call resumeManualProcess for session " + sessionId);
    ServiceResult result = this.sws4bpel.resumeManualProcess(sessionId, selectedServiceId);
    log.debug("----------> InReturned data: " + result.toString());
    return result;
}

@WebMethod(operationName = "LookupSession")
public Session performSessionLookup(Integer sessionId) {
    log.debug("----------> Web Service ready to call performSessionLookup for session " + sessionId);
    Session session = this.sws4bpel.performSessionLookup(sessionId);
    log.debug("----------> InReturned data: " + session.toString());
    return session;
}

@WebMethod(operationName = "PerformInvocation")
public ServiceResult performInvocation(ServiceQuery serviceQuery, Service service) {
    log.debug("----------> InReceived data: " + serviceQuery.toString() + ", " + service.toString());
    ServiceResult result = this.sws4bpel.performInvocation(serviceQuery, service);
    log.debug("----------> InReturned data: " + result.toString());
    return result;
}

@WebMethod(operationName = "PerformServiceSelection")
public Service performServiceSelection(ServiceQuery serviceQuery, @WebParam(name = "ServiceList") Collection<Service> services) {
    log.debug("----------> Web Service ready to call performServiceSelection");
    Service selected = this.sws4bpel.performServiceSelection(serviceQuery, services);
    log.debug("----------> InReturned data: " + selected.toString());
    return selected;
}
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

<import resource="classpath:org/codehaus/xfire/spring/xfire.xml"/>

<bean id="serviceBean" class="no.ntnu.idi.wisemod.sws4bpel.ws.SWS4BPELWebService">
  <property name="sws4bpel">
    <bean class="no.ntnu.idi.wisemod.sws4bpel.core.SWS4BPELServiceImpl"/>
  </property>
</bean>

<bean id="webAnnotations" class="org.codehaus.xfire.annotations.jsr181.Jsr181WebAnnotations"/>

<bean id="jsr181" class="org.codehaus.xfire.annotations.AnnotationServiceFactory">
  <constructor-arg ref="webAnnotations"/>
</bean>

<bean id="sws4bpelService" class="org.codehaus.xfire.spring.ServiceBean">
  <property name="serviceBean" ref="serviceBean"/>
  <property name="serviceFactory" ref="jsr181"/>
</bean>
</beans>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/spring-xfire.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>XFireServlet</servlet-name>
    <display-name>XFire Servlet</display-name>
    <servlet-class>org.codehaus.xfire.spring.XFireSpringServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>