NTNU
Norwegian University of
Science and Technology

Semantic Web Services: an Evaluation of a Framework Implementation

Øyvind Skytøen

Master of Science in Computer Science
Submission date: June 2008
Supervisor: Guttorm Sindre, IDI

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

Problem Description

The amount of available information on the web is growing rapidly along with the available Web services. With the growing amount of information comes an increasing demand for better searching facilities for both humans and machines. To improve the web with machine understandable content can facilitate new intelligent software that roam the web performing various tasks for users. This is the main goal of the Semantic Web. With the Semantic Web follows increased focus on Semantic Web Services (SWS).
A lot of research has been done on adding semantic annotation to Web services to create Semantic Web Services. Several frameworks have emerged from these studies. Some of these frameworks have working implementations. It would be interesting to study one of these implementations to determine if it is able to realize the goal of Semantic Web Services.
The framework should be evaluated according to how it is able to complete a scenario that includes the creation and execution of a Semantic Web Service. It should also be evaluated on how easy this can be done, and on the available documentation.

Assignment given: 14. January 2008
Supervisor: Guttorm Sindre, IDI

Abstract

The World Wide Web has become a vast and often chaotic source of all sorts of information and services. Computers have become the tool of the modern knowledge worker, and the Web constitutes both office and library. The current tools of the Web are unable to utilize its full potential, which limits the efficiency of the knowledge workers. The goal of the Semantic Web, and Semantic Web Services, is to solve this problem by introducing an evolution of the Web that is understandable for machines and humans.

The objective of this master thesis was to extend the evaluation in my project on Semantic Web Services frameworks from fall 2007, by evaluating a framework implementation. The evaluation was to focus on the framework implementation's ability to realize the goals of Semantic Web Services, how easy a Semantic Web Service could be created for the given implementation, and to evaluate the available documentation and tools. The goal of the evaluation was to come up with suggestions for improvements for the framework implementation.

The WSMO framework and its WSMX implementation was chosen as subject of the evaluation, based on the evaluation from the first project. The framework was evaluated by implementing a Semantic Web Service from a constructed scenario. The development of this service resulted in positive and negative experiences with the WSMX implementation, experiences that were used in the evaluation. The evaluation focused on WSMX, the available documentation, and the two tools WSMT and WSMO Studio.

The results of the evaluation were suggestions for improvements for WSMX, the documentation, and the tools. By making the changes and additions that were suggested, I believe that the development of Semantic Web Services for the WSMX implementation of the WSMO framework could be made easier.

Preface

This Master thesis builds on my evaluation of four Semantic Web Services frameworks performed during fall 2007 [28], and concludes my MSc in Computer Science at the Department of Computer and Information Science (IDI) at the Norwegian University of Technology and Science (NTNU).

I would like to thank my teaching supervisor Guttorm Sindre for guidance and sharing of expertise during this project.

Øyvind Skytøen Trondheim, June 8, 2008

Contents

List of Figures

Appendix

1 Introduction

Since the beginning of the Internet in the late 1960's [24] and the introduction of the World Wide Web (WWW) at CERN in 1991 [23], the WWW has come a long way. The WWW now plays an important role in the daily lives of millions of people around the world [15]. The web is now a vast collection of human readable information. After the introduction of Web services and the increasing focus on Service-Oriented Architecture (SOA) [18], we see an increasing number of resources that would be beneficial for machines and applications. The web services in their present form can be considered as isolated islands of functionality. Human interaction for discovery and integration into external applications is needed to access this functionality [14].

The desired functionality is an autmatic bridging of the islands, enabling agents to solve more complex problems by combining services. By adding rich formal descriptions of the capabilities of a Web service, we can enable automation of the composition, discovery, dynamic binding, and invocation of services within an open environment [8].

1.1 Motivation

The amount of available information on the web is growing rapidly along with the available Web services [15]. Along with the growing amount of information comes an increasing demand for better searching facilities for both humans and machines. To improve the web with machine understandable content can facilitate new intelligent software that roam the web performing various tasks for users. This is the main goal of the Semantic Web [5]. With the Semantic Web follows increased focus on Semantic Web Services (SWS) [26].

Considerable effort has been put in the study of annotating Web services with semantic content to create Semantic Web Services in the recent years. This research has produced several Semantic Web Services Frameworks. Among these frameworks are the METEOR-S, WSMO, IRS-III and OWL-S. In [28] these frameworks were evaluated to try to determine if they are able to realise the goals of Semantic Web Services. This evaluation did not include any studies of the frameworks available implementations, so it would be interesting to take a closer look at one of these, in order to further evaluate the chosen framework.

1.2 Problem Definition

Because of the growing amount of information available on the Web, the Semantic Web has emerged. With the Semantic Web follows an increased focus on Semantic Web Services. The goal of SWS is to provide the means to facilitate automation of tasks like composition, discovery, dynamic binding, and invocation of services [8].

This master thesis should build on the evaluation in [28], by evaluating the implementation of one of the frameworks considered there. The first step of the process would be to use the results from [28] to choose one of the frameworks as the subject of further evaluation. This framework implementation should then be evaluated on its success

of providing an environment for SWS execution, and how easy it is to build and run a SWS on it. The documentation and available tools for development should also be considered in the evaluation. The concrete goals of the project are:

1. Find one framework suitable for an evaluation of its implementation, based on the results from [28]
2. Describe the chosen framework and its implementation
3. Develop a Semantic Web Service to be run on the given implementation
4. Evaluate the framework implementation based on the success of running the implemented Semantic Web Service, and how easy this was. The available documentation and tools should also be evaluated.
5. Suggest improvements for the framework implementation, based on findings from the evaluation

The results from this master thesis should be an evaluation of the chosen framework, with suggestions for future improvements. The deliveries from the project should be a report with this evaluation, and a running Semantic Web Service implementation.

1.2.1 Scope

The evaluation performed in this master thesis should only consider one framework. The Semantic Web Service that is to be implemented should be kept simple enough to be implemented within the given time constraint of 20 weeks, including the writing of this report. It will not be possible to perform an in depth study of the quality of the framework implementation code, as this task would not be possible to carry out within the allotted timeframe.

1.3 Project Context

The work documented in this report is related to the Web Information Service Modeling (WISEMOD) project [2], funded by the Research Council of Norway (NFR).

1.4 Method

The method chosen to form the basis of the evaluation in this master thesis is an exploratory case study. An exploratory case study is usually triggered by a ”what” question [32], like ”what changes could be made to a framework implementation to make it better”. Such a case study is often conducted as a pilot study before implementing a large scale investigation. This fits well for this project, since it is only able to tap the surface of the framework implementation. A larger case study is needed if the entire framework is to be further evaluated.

The case study will be the development of a Semantic Web Service for the chosen framework implementation. A scenario will be defined that will be used as the basis for this development. The scenario should be designed so that as many aspects as possible of the framework implementation will be in use (limited by the scope of this project). This developed implementation will form the basis for the evaluation.

2 Semantic Web Services Frameworks

In this section we present four framworks for Semantic Web Services. The four frameworks were identified in [28] as the most promissing to provide what is needed to fulfill the goals of Semantic Web Services. These four are METEOR-S, Web Service Modeling Ontology (WSMO), Internet Reasoning Service (IRS) and Web Ontology Language for Services (OWL-S). The results of [28] is also summarized, and one framwork is chosen for further study.

2.1 METEOR-S

METEOR-S is the follow on project to the METEOR[1] project at the LSDIS[2] Lab; University of Georgia. The METEOR project focused on workflow management techniques for transactional workflows, and deals with the complete lifecycle of semantic and dynamic Web processes [30]. The main focus is the automation of the configuration of Web processes.

The METEOR-S framework allows dynamic configuration and re-configuration of Web processes. It uses semantic descriptions of the services and process constraints to achieve this. There are four components that together constitutes METEOR-S [30]: the METEOR-S Semantic Web Service Annotation Framework (MWSAF), the METEOR-S Web Service Composition Framework (MWSCF), the METEOR-S Web Service Discovery Infrastructure (MWSDI), and the METEOR-S Composer.

The MWSAF is a tool for semi-automatic annotation of existing Web services described in WSDL. Semi-automatic means that the MWSAF tools allow existing WSDL documents to be mapped to ontology concepts using automated reasoning, and manual control. The MWSCF is a tool for designing abstract processes. The abstract processes are designed using the WS-BPEL[3] workflow standard. The MWSDI enables the publishing of semantically annotated Web Services and allows users to discover services. The METEOR-S Composer is a tool for orchestration and composition of Web services.

2.1.1 Architecture

The high level architecture of METEOR-S has three main components: the process designer, the configuration module, and the execution environment. The architecture is shown in Figure 1.

The Process Designer is a module of MWSCF where abstract processes represented in WS-BPEL can be designed [31]. It is implemented in Java and uses WSDL4J [3] to process WSDL files.

[1] METEOR-S - Managing End-To-End OpeRations
[2] LSDIS - Large Scale Distributed Information Systems
[3] WS-BPEL - Web Services Business Process Execution Language

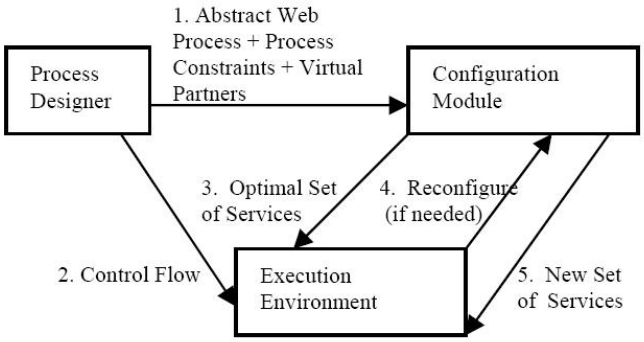

Figure 1: METEOR-S Architecture Overview [30]

The Configuration Module is responsible for dynamically choosing and binding services for Web processes. The selection is done on the basis of semantic descriptions of the Web services, and process constraints. The MWSAF tool is used together with WSDL-S [4] to add annotation to Web services.

The Execution Environment consists of a logical layer over a Web process execution engine. The execution engine uses proxies for each individual partner of the process. Run-time and deployment time binding is supported by the possibilty of changing the service bound to the proxies. A Web process execution engine invokes a particular operation of a service to execute each part of a process.

2.1.2 Implementation

The METEOR-S framework is currently in version 0.8 and has a couple of finished tools [1]. METEOR-S provides eclipse plug-ins for developing WSDL-S documents and processes, as well as a couple of ontologies for WSDL-S annotations. The MWSAF is currently the only METEOR-S component available for download. The remaining components are yet to be released.

2.2 Web Service Modeling Ontology

The Web Service Modeling Ontology (WSMO) is a project undertaken by the WSMO Working group at the SDK Cluster. WSMO is a framework providing ontological specifications for a set of core elements of Semantic Web Services [27]. The goal of WSMO is to describe all relevant aspects related to services that are accessible through a Web Service interface, and it aims to enable automation of the tasks in the integration of Web Services (e.g. discovery, selection, composition, mediation, execution, monitoring, etc.).

WSMO has two subprojects:

1. **The Web Service Modeling Language (WSML)** is a language for description of WSMO Web services, and can be used for the precise specification of the single elements in the WSMO framework [7].

2. **The Web Service Execution Environment (WSMX)** is a comprehensive execution environment for Web services, and is the reference implementation of WSMO [17].

WSMO defines four top level elements as the main concepts which have to be described to define Semantic Web Services. The four elements are: Ontologies, Web Services, Goals, and Mediators. Each of the top level elements has a set of new elements describing it.

Ontologies are a central part if the WSMO framework, and provides the terminology used by the other core elements to describe the relevant aspects of the spesific domain [27]. To be able to solve mismatches between different ontologies, each ontology is associated with a *ooMediator*.

Web Services is an element of WSMO providing a conceptual model for the description of all aspects of a Web service [27]. By providing a model for Web services that is unambiguous and has well-defined semantics, WSMO Web services can be processed and interpreted by computers without human intervention. This enables automation of tasks like discovery, selection, execution, monitoring, composition and mediation.

Goals in WSMO describe the user's desires. They provide the specification of the objectives from the requester to the Web service [27]. The provided goals are resolved by selecting from the available Web services that can fulfill such goals at the highest satisfaction.

Mediators in WSMO are used to resolve heterogeneities between the different components [27]. There are four types of mediators: *OO Mediators*, *GG Mediators*, *WG Mediators*, and *WW Mediators*. The mediators connect two Ontologies, two Goals, a Web service and a Goal, and two Web services respectively.

2.2.1 Architecture

The architecture of WSMO is concretised through WSMX, and its main purpose is to provide a reference implementation of WSMO [17]. WSMX can be used by both Web service providers and requesters. Providers may register their services using WSMX in order to make them available to the consumers. Requesters can use WSMX to find the Web services that suit their needs and invoke them. To realize this, WSMX in itself is a Web service. The architecture of WSMX can be seen in Figure 2.

The *Compiler* validates and compiles data describing a Web service, and store it in the repository. The *Matchmaker* matches the goals from the requester against the capability of the Web services in the *Service Repository* and returns a set of Web services. The *Selector* chooses the Web Service that best suits the requester's preferences. Eventual mismatches are handled by the *Data Mediator*.

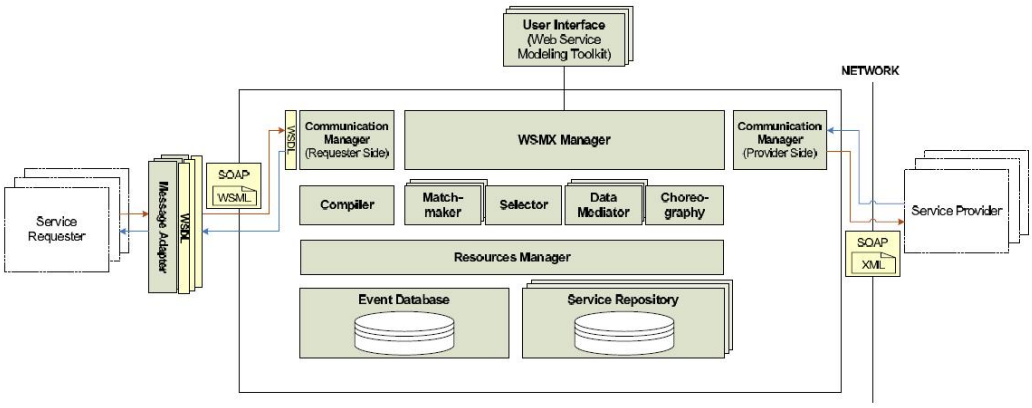

Figure 2: Simplified WSMX Architecture [17]

The *Communication Manager* invokes the selected Web services. The *Choreography Engine* is used to link the communication patterns of the requester and the Web service. The execution semantics of the system is implemented in the *WSMX Manager*. The *Resource Manager* offers an abstraction layer to the persistence layer.

2.2.2 Implementation

An implementation of WSMX is provided through SourceForge [4]. Both source code and binary is available for download. The Web Services Modeling Toolkit (WSMT) is a framework for the rapid deployment of graphical administrative tools, which can be used with WSMO, WSML and WSMX. The implementation is available through the WSMX project on SourceForge.

2.3 Internet Reasoning Service

The Internet Reasoning Service (IRS) is an ongoing project at the Knowledge Media Institute at the Open University [21]. The overall goal of the IRS project is to support automated or semi-automated construction of semantically enhanced systems over the internet. IRS is a Semantic Web Services framework, which allows applications to semantically describe and execute Web services. IRS-III is the current version of IRS. In this version the WSMO ontology (described in Section 2.2) has been incorporated and extended so that the implemented infrastructure allows the description, publication and execution of Semantic Web Services [9]. IRS-III provides the representational and reasoning mechanisms for implementing the WSMO meta-model to describe Web Services. It also provides an execution environment.

2.3.1 Architecture

IRS-III is based on a distributed architecture composed of the IRS-III server, the publishing platforms and clients which communicate through the SOAP protocol. These

[4] http://sourceforge.net/projects/wsmx

three components together constitute the IRS-III framework pictured in Figure 3.

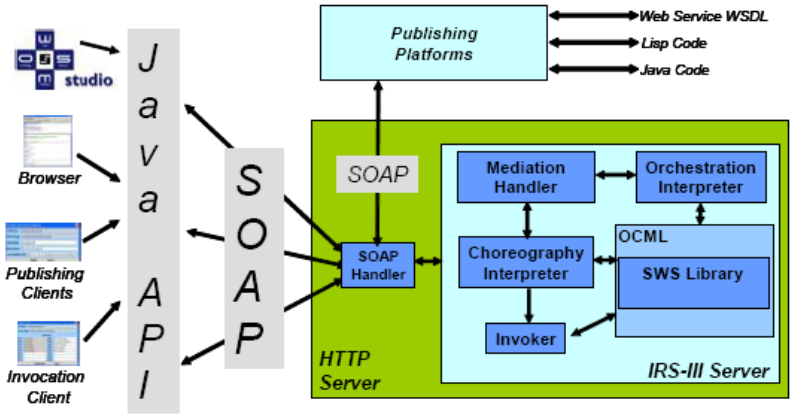

Figure 3: The IRS-III Framework [9]

The IRS-III server handles ontology management and the execution of knowledge models defined for WSMO. The client applications sends SOAP requests to the server for creating and editing WSMO descriptions of goals, web services and mediators. The server uses an HTTP server written in Lisp that has been extended with a SOAP handler. The publishing platforms allow providers of services to attach semantic descriptions to their deployed services. It also provides handlers to invoke services in a specific language and platform.

2.3.2 Implementation

The IRS-III server is written in Lisp and is available as an executable file. The publishing platforms are delivered as Java Web applications, and client applications use the Java API. A IRS-III distribution is available as a Java client with appurtenant Javadoc through the IRS Web page[5]. WebOnto is a tool for visualizing and editing IRS-III ontologies in OCML[6].

2.4 Web Ontology Language for Services

Web Ontology Language for Services (OWL-S) is an OWL-based Web Service Ontology developed by the Semantic Web Services arm of the DAML program [7]. The latest version of OWL-S is version 1.2, which is labeled as a pre-release. Because of this, release 1.1 is considered here. OWL-S aims to deliver semantics for Web services in order to not only represent content with semantics, but also allow one to effect some action or change in the world, such as the sale of a product or the control of a physical device [25].

[5] http://kmi.open.ac.uk/projects/irs/
[6] http://anfield.open.ac.uk:3000/webonto
[7] http://www.daml.org

2.4.1 The Upper Ontology of Services

The structuring of the ontology of services is motivated by the need to provide three types of knowledge about a service. These three are: *what does the service provide for prospective clients*, *how is it used*, and *how does one interact with it*. To answer these questions each Service class presents a ServiceProfile, a ServiceGrounding and a ServiceModel. These classes form the top level of the service ontology as seen in Figure 4

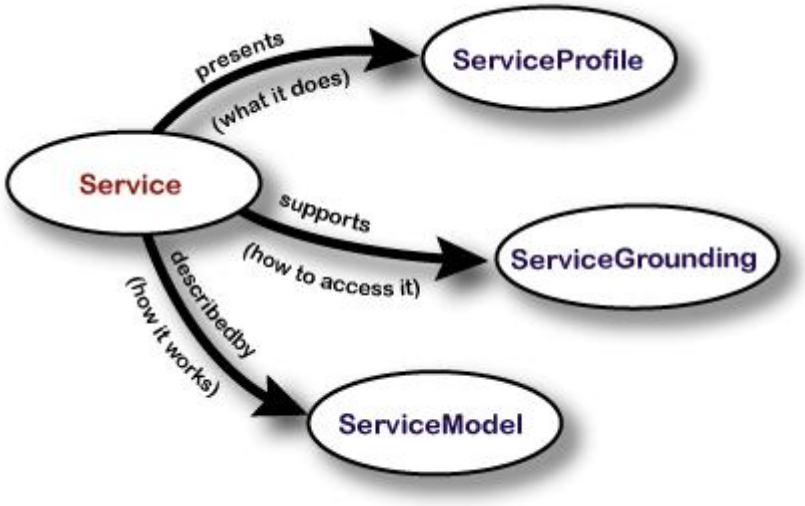

Figure 4: Top Level of the Service Ontology [25]

The class Service provides an organizational point of reference for a declared Web service. One instance of this class exists for each distinct service. The Service class has three properties: *presents*, *describedBy*, and *supports*. The ranges for these properties are the classes ServiceProfile, ServiceModel and ServiceGrounding respectively.

The Service Profile provides the information needed for an agent to discover a service. The description includes what is accomplished by the service, limitations on service applicability and quality of service, and requirements that the service requester must satisfy to use the service successfully.

The Service Model tells a client how to use the service by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and the step by step process leading to those outcomes. The service model in OWL-S is modeled as a *process*. A process is not a program to be executed, but a specification of the ways a client may interact with a service.

The Service Grounding specifies the details of how an agent can access a service. A grounding usually specify a communication protocol, message formats, and other service-specific details like port numbers.

2.4.2 implementation

Not all parts of OWL-S has been implemented so far. There have been released OWL-files for the Upper Ontology for Services, and other relevant ontologies used by the Upper Ontology [10]. Files with semantic for Web services are aslo available. Some tools have also been developed to support OWL-S: the OWL-S Protégé-based Editor can be used to create and maintain OWL-S service descriptions [8], the OWL-S Editor can be used to create OWL-S descriptions for a Web service [9], the ASSAM Web Service Annotator is used to annotate Web services [10], the Semantic Web Service Composer is a semantic matching and composition engine [11], and the OWL-S Matcher computes different degrees of matches for elements of OWL-S descriptions [12].

2.5 Evaluation

In [28] the four frameworks were compared and evaluated on their approaches in supplying Web services with semantic description to support automatic discovery, composition and execution. The availability of implementations were also evaluated. A summary of the findings are given here.

2.5.1 Approaches Used in the Framworks

[8] describes a set of activities that a Semantic Web Service might perform to fulfill the vision of supporting automation of the use of Web services. The supporting architecture of these activities is also described. [28] identified in what part of the architecture of each framework these activities were supported. The results are shown in Table 1.

The table shows that OWL-S is lacking support for some of the activities, like for instance publishing. The other three frameworks describe how all the activities can be realized, and are considered more complete [28].

2.5.2 Available Implementations

Only two of the frameworks have working implementations available for download. Both WSMO (WSMX) and IRS-III provides a server which can be set up on any computer. Only WSMO has made the source code for this implementation available. For METEOR-S detailed descriptions of some of the tools are available, but no implementations have been made available for download. Implementation of some tools exists for OWL-S, but these do not provide a full reference implementation. The lack of implementations for METEOR-S and OWL-S speaks in favor of WSMO and IRS.

[8] http://owlseditor.semwebcentral.org/
[9] http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html
[10] http://moguntia.ucd.ie/projects/annotator/download
[11] http://alphaworks.ibm.com/tech/ettk
[12] http://owlsm.projects.semwebcentral.org/

Table 1: Summary of SWS activities [28]

SWS Activity	METEOR-S	WSMO	IRS-III	OWL-S
Publishing	Process Designer	WSMO Editor/Service Repository	Publishing Client/ Handler	Not detailed
Discovery	Configuration Module	Matchmaker	Mediation Handler	Matcher
Composition	Configuration Module	Matchmaker	Mediation Handler	Not detailed
Selection	Configuration Module	Selector	Mediation Handler	Matcher
Invocation	Execution Engine	Communication Manager	Invocation Handler/ Publishing Platforms	Not detailed
Deployment	MWSAF	Matchmaker	Publishing Platforms	Not detailed
Ontology Management	MWSAF	WSMO Editor	Mediation Handler	OWL-S Editors

2.5.3 Choice of Framework for Further Evaluation

The lack of available implementations for METEOR-S and OWL-S makes it hard to further evaluate these frameworks, since this is the implementation is the next step in the evaluation. Both WSMO and IRS have available implementations, and are thus possible choices for further evaluation. The fact that the source code is available for WSMO, makes it favorable over IRS. If a change in the software is needed, it could be done with WSMO but not with IRS. The WSMX implementation of WSMO is also updated continually, which is not the case for IRS. WSMO is thus chosen for further evaluation and study.

3 Web Service Modeling Ontology

As stated in Section 2.2, the Web Service Modeling Ontology is a project undertaken by the WSMO Working group at the SDK Cluster. It describes a framework providing ontological specifications for a set of core elements of Semantic Web Services [27]. In this section the two subprojects of WSMO, WSML and WSMX, will be described in more detail.

3.1 Web Service Modeling Language

The Web Service Modeling Language (WSML) is a language for description of WSMO Web services, and can be used for the precise specification of the single elements in the WSMO framework [6]. The designers of WSML wanted to create a framework that brings together Web technologies and logical language paradigms in order to enable the description of Semantic Web Services. The starting points for WSML are Description Logics, Logic Programming and F-Logic. This is useful for enabling automatic reasoning around WSMO concepts, and to provide efficient mediation. The latest stable version of WSML is version v0.21, which was released in October 2005. Version v0.3 is currently under development [29].

3.1.1 WSML Variants

WSML exists in five variants: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule, and WSML-Full [6]. The five variants and their relations are shown in Figure 5.

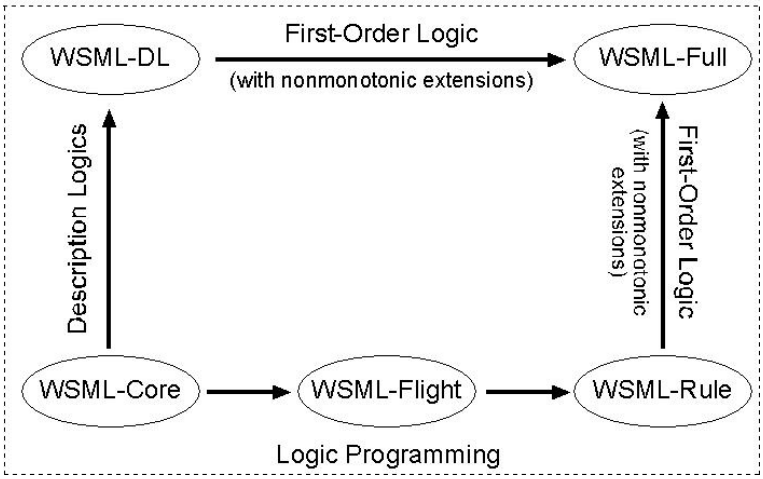

Figure 5: WSML variants [6]

WSML-Core corresponds with the intersection of Description Logic and Horn Logic, extended with datatype support in order to be useful in practical applications. WSML-DL extends WSML-Core to an expressive Description Logic. WSML-Flight extends WSML-Core in the direction of Logic Programming, and has a rich set of modeling primitives for modeling different aspects of attributes, such as value constraints and

integrity constraints. WSML-Rule extends WSML-Flight to a fully-fledged Logic Programming language, including function symbols. WSML-Full unifies the Description Logic and Logic Programming paradigms.

3.1.2 WSML Syntax

The five variants of WSML are specified in terms of a human-readable syntax with keywords similar to the elements of the WSMO conceptual model [6]. In addition, XML and RDF exchange syntaxes, as well as a mapping between WSML ontologies and OWL ontologies for interoperability with OWL-based applications are provided.The WSML syntax consists of two major parts: the conceptual syntax and the logical expression syntax. The conceptual syntax is used for the modeling of ontologies, goals, web services and mediators. The logical expressions are used to refine these definitions using a logical language.

The conceptual syntax for WSML has a frame-like style, meaning that information about a class and its attributes, a relation and its parameters and an instance and its attribute values is specified in one large syntactic construct. A WSML specification is separated into two parts. The first part provides meta-information about the specification like; WSML variant identification, namespace references, non-functional properties (annotations), import of ontologies, references to mediators used and the type of the specification. The second part consists of elements such as concepts, attributes, relations (in the case of an ontology specification), capability, interfaces (in the case of a goal or web service specification), etc.

Ontology Specifications An ontology specification is identified by the *ontology* keyword optionally followed by an IRI[13] which serves as the identifier of the ontology. The next element is a header with non functional properties, imported ontologies and used mediators. After the header follows different ontology elements:

- **Concepts -** the different concepts of an ontology (e.g. human or animal). It consists of an identifier, a list of *subConceptOf*, non functional properties and a list of attributes.
- **Relations -** the relations between two or more concepts.
- **Instances -** the instances of the concepts. It consists of an identifier, a list of *meberOf*, non functional properties and a list of attribute values.
- **Relationinstances -** the instance of a relation.
- **Axioms -** the axioms of an ontology.

Capability Specifications A capability constitutes a formal description of the functionality requested from or provided by a web service. A capability specification starts with the *capability* keyword, which is followed by an identifier (name) and a

[13]IRI- Internationalized Resource Identifier

header. The next element is a list of shared variables. After the shared variables follows:

- **Preconditions -** describe conditions on the input of the service.
- **Postconditions -** describe the relation between the input and the output of the service.
- **Assumptions -** describe what must hold (but cannot be checked beforehand) of the state of the world for the web service to be able to execute succesfully.
- **Effects -** describe real-world effects of the execution of the web service which are not reflected in the output.

Interface Specifications An interface describes the input and output requested from or provided by a web service. Multiple interfaces are possible. An interface specification starts with the *interface* keyword, which is followed by an identifier (name) and a header. After the header follow choreographies and orchestrations.

A choreography deals with interactions of the Web service from the client's perspective. The state-based mechanism for describing WSMO choreography interfaces is inspired from the Abstract State Machine (ASM) methodology [13]. An ASM is used to abstractly describe the behavior of a service or the behavior of a client when interacting with the service. A choreography consists of non functional properties, a state signature, a state and transition rules which are described below.

- **State Signature -** defines the state ontology used by the service together with the definition of the types of modes the concepts and relations may have. The five possible modes in a state signature are: static, in, out, shared, and controlled. A grounding mechanism must be provided for each item of the modes in, out and shared. The grounding mechanism implements read/write access for the environment and must be provided.
- **State -** of the choreography is defined as a set of ground facts.
- **Transition Rules -** express changes of states by changing the set of instances (adding, removing and updating instances to the signature ontology).

The details on orchestrations have not been described by neither the WSML or WSMO working groups at the time of writing.

Goal Specifications A goal specification is identified by the *goal* keyword optionally followed by an IRI which serves as the identifier of the goal. The next element is a header. After the header follows the capability and the interfaces which are described above.

Web Service Specifications A Web service specification is identified by the *webservice* keyword optionally followed by an IRI which serves as the identifier of the Web service. The next element is a header. After the header follows the capability and the interfaces which are described above.

Mediator Specifications Four types of mediators are allowed in WSML: mediators between ontologies, mediators between web services, mediators between goals, and mediators between Web services and goals. The respective keywords for the mediators are: *ooMediator*, *wwMediator*, *ggMediator*, and *wgMediator*. A mediator is identified by one of these keywords optionally followed by an IRI which serves as the identifier of the mediator. The remaining elements are sources, targets and services used.

3.2 Web Service Execution Environment

WSMX is an execution environment for Semantic Web Services, and is the reference implementation for a semantic execution environment (SEE) middleware based on the conceptual model provided by WSMO [19]. The execution environment is implemented in Java, while the Web Service Modeling Language (see Section 3.1) is used as the underlying language for the semantic descriptions. WSML-Flight is currently the only supported variant of WSML in the WSMX implementation. Source code and binary distribution of WSMX is available from SourceForge.net [14]. Details of the architecture of WSMX are described in the subsequent section.

3.2.1 WSMX Architecture

WSMX is based on Service Oriented Architecture (SOA) principles. Self-contained components provide different functionality to the system, and acts as middleware service providers. Communication between the components are event-based, and uses a publish-subscribe mechanism. The high level architecture of WSMX can be seen in Figure 6.

The Core is the central component. It provides middleware frameworks functionality such as finding and loading components, handling the messaging between components, and defining paths of execution. The type of components in the Integration API is also defined by the Core.

The other components are decoupled from each other by wrappers. This is done to enable easy plug in of additional components. The components can also be switched with any third-party component that realises the same functionality. The different components are described below.

- **Core:** Integrates all the other components by providing a middleware framework functionality.
- **Coreography:** Resolves process heterogeneity (in terms of communication mismatches) between service requester and provider.

[14] http://sourceforge.net/projects/wsmx/

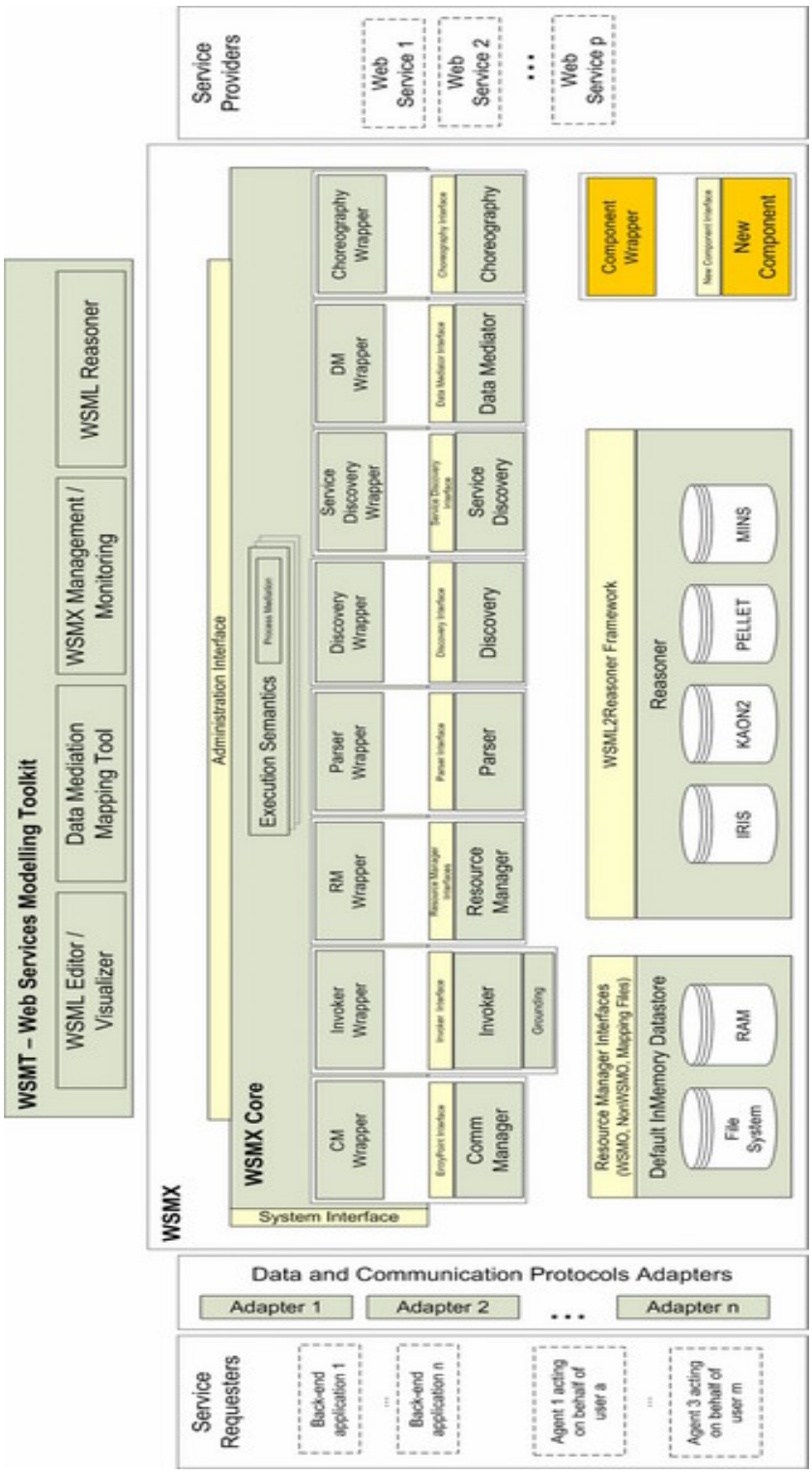

Figure 6: High Level WSMX Architecture [19]

- **Communication Manager:** Entry point to the system for external entities that want to consume WSMX functionality.
- **Data Mediator:** Resolves data heterogeneity that can appear during discovery, composition, selection, or invocation of web services by transforming instances of the ontologies known to one of the parties to instances of the ontologies known to the respective other party.
- **Invoker:** Handles communication between WSMX and the external SOAP-based Web services. This includes lowering/lifting to/from WSML when communicating XML-based SOAP messages.
- **Orchestration:** Defines how the overall functionality of a service is achieved by the cooperation of other services in order to resolve process heterogeneity.
- **Parser:** Performs syntactic validity checks of WSML documents and converts it to an in memory representation.
- **Resource Manager:** In-memory implementation which uses the file system for initial loading at startup.
- **Service Discovery:** Provides instance-based service discovery with service contracting, and QoS discovery based on nonfunctional properties.
- **Web Service Discovery:** Given a goal definition, this component finds Web service definitions in the repository which can fulfill that goal.

3.2.2 WSMX Access and Administration

The functionality of WSMX is accessible through a series of entry points. The most important entry points are:

- **achieveGoal:** Starts the AchieveGoal execution semantics. Input: goal to achieve incl. input data (instances). Returns result instances.
- **discoverWebServices:** Discover Web services based on goal. Input: goal. Returns discovered web services.
- **invokeWebService:** Invoke a Web Service. Input: Web service and goal with instances. Returns result instances.

The operations are available through the Server Administration GUI or via Web services as explained below.

Web Service Access The WSDL of the web service exposing the WSMX entry points and additional operations is available at http://localhost:8050/axis/services/WSMXEntryPoints?wsdl. The invocation of a Web service can be done programmatically or by using WSMT.

The Web services that are created to provide access points for requesters that need adaption to communicate with WSMX, can be found in a service list at http://localhost:8050/axis/services/listServices.

Server Administration GUI The Server Administration GUI contains information about the running WSMX instance and facilitates basic administration tasks. It is available at http://localhost:8080 (the port is specified in the configuration).

The server view offers important functionality from different services exposed as MBeans [15]. The server view can be seen in Figure 7. Examples of functionality include:

- WSMX shutdown using the core/WSMXKernel MBean
- WSMX entry points via the components/Communication Manager MBean

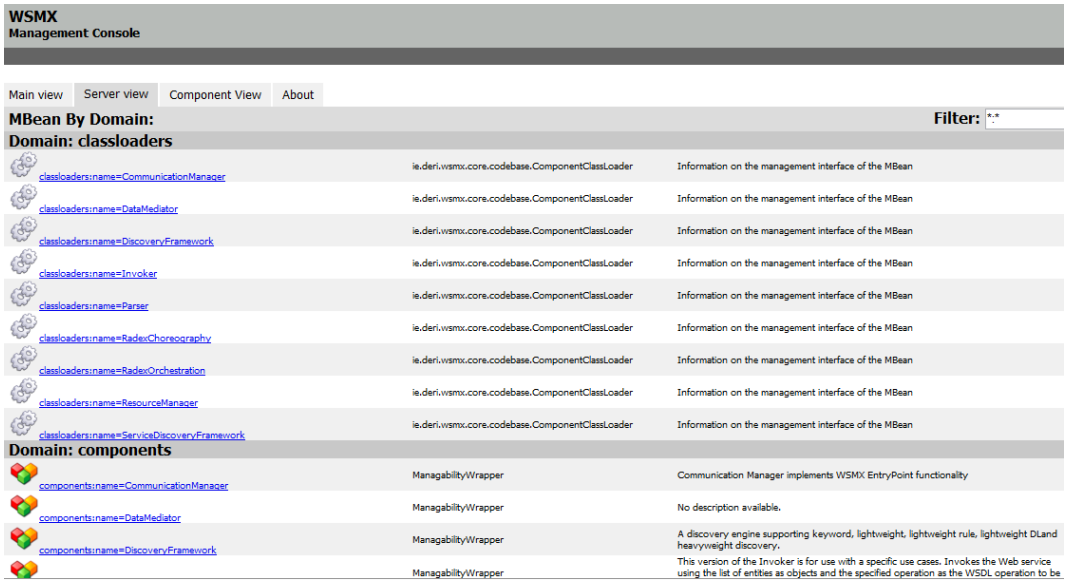

Figure 7: Server Administration GUI, Management Console

Web Services Modeling Toolkit The Web Services Modeling Toolkit (WSMT) is an Integrated Development Environment (IDE) for Semantic Web Services implemented in the Eclipse framework. WSMT aims to aid developers of Semantic Web Services in the WSMO paradigm, by providing a seamless set of tools to improve their productivity when working with WSMO, WSML and WSMX [20].

The three main areas of the functionality of WSMT is the engineering of WSMO descriptions, creation of mediation mappings and interfacing with execution environments. WSMT is broken up into three perspectives that offer this functionality:

[15] A JMX MBean (managed bean) is a Java Object that represents a manageable resource, such as an application, a service, a component, or a device [19]

- **The WSML Perspective:** This perspective is used to create ontology descriptions. In addition to a text editor for creating ontology descriptions a WSML Visualiser and a WSML Form based Editor is also provided. Validation of WSML is handled by WSMO4J [16], which gives instant feedback on errors.
- **The Mapping Perspective:** This perspective is used to create mapping files for mediation between ontologies. All mappings within WSMT are in the Abstract Mapping Language (AML). An AML Text Editor is provided together with a View Based Editor for more graphical creation of mappings. Validation of AML artifacts is also available.
- **The SEE Perspective:** This perspective is used to interact with the WSMX server. The Web service interfaces of WSMX are accessible through this perspective and the SOAP Message View. All descriptions currently stored on the server are shown and can be downloaded into the workspace.

WSMT offer more functionality than what is described in the list above, but these are the most important.

[16] WSMO4J - WSMO for Java: an object model for manipulating WSMO descriptions. Also provides validators for each WSML variant.

4 Implementation

In this section we present the implementation that was carried out as a part of the evaluation of the WSMO framework. The first section describes the Semantic Web Service scenario that has been implemented. In the second section the different tools that were used during the implementation is described. The third section gives details of the actual implementation of the different parts, and the last section describes the final solution.

4.1 Semantic Web Service Scenario

In this section we present a scenario that will be implemented and run on the WSMX server. The scenario is created specifically for this project, and concerns an online search for book prices. The choice of this scenario is partly based on the availability of free existing Web services, because the implementation of one or more Web services is beyond the scope of this project. It is also chosen because it gives the opportunity to test the most important and basic parts of WSMX.

4.1.1 The Scenario

The scenario, as already stated, concerns the search for book prices. Abundant Technologies [17] offers some free Web services that are open for anyone to use. One of these is a Web service which allows the user to search for the price of books at Barnes & Noble (an American bookseller company, sells books both online[18] and in shops), based on the ISBN [19] identification of the book. The WSDL describing the Web service is available at http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl. The scenario can be seen in Figure 8. From a users point of view this scenario acts as follows:

- The user desires to buy a book, but want to check the price first.
- The user finds the ISBN of the book he wants to know the price of.
- The user sends a request to the Semantic Web Service offered by the WSMX server with the ISBN of the book.
- The user receives a reply from the SWS with the price of the book.

This scenario is very simple, and the use of a Semantic Web Service is actually not needed in this case. The user could of course just send the request straight to the Abundant Technologies Web service. The reason for choosing this scenario is to keep the implementation within the scope of the project, and because it uses the most important and basic parts of WSMX. To be able to fulfill this scenario, WSMO ontology

[17] http://www.abundanttech.com/
[18] http://www.barnesandnoble.com/
[19] ISBN - International Standard Book Number

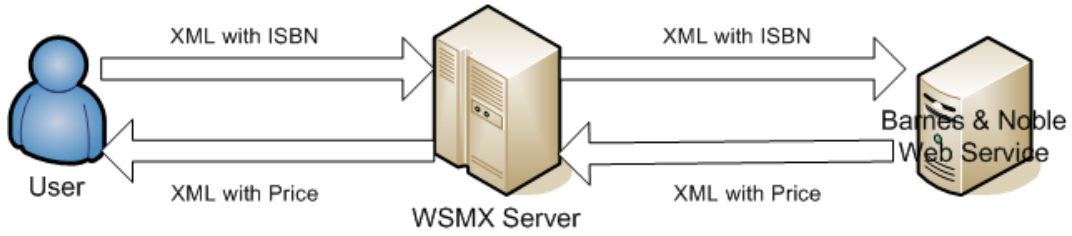

Figure 8: SWS Scenario

description for an ontology, a goal and a Web service must be created. Adapters must also be created to be able to communicate with the Web service.

4.1.2 Possible extensions

As stated in the previous section, mediators will not be needed in the given scenario. If the scenario is extended to provide the lowest price from a set of booksellers, the use of mediators would be needed. WSMO ontology descriptions for the different Web services would then be needed, each with a possible related ontology if the input is not the same. Mediation between the different ontologies would be needed to interpret the results from the different services. This extension would also result in more use of the logic programming possibilities of WSML to rank the prices returned from the Web services.

If the appropriate Web services are available, the scenario could also be extended to include the actual purchase of the book for the lowest available price. The online shopping site Amazon offers such services through their Amazon Associates Web Service [20].

4.2 Tools

Different tools were used during the implementation. Some of these are tools that have been made to support WSMO and WSMX, and others are more general. The different tools are used for programming, writing ontology descriptions and Web service interaction. Details on each are given subsequently.

4.2.1 The Eclipse Platform

The Eclipse Platform [21] is an open source project, and provides an excellent IDE [22] for Java development. The implementation of WSMX was easily downloaded from a SVN [23] repository on SourceForge into Eclipse after installing the Subclipse plug-in [24]. A wide variety of extensions to Eclipse are available as plug-ins, and both WSMT WSMO

[20] http://www.amazon.com
[21] http://www.eclipse.org/
[22] IDE - Integrated Development Environment
[23] SVN - Subversion
[24] http://subversion.tigris.org/

Studio, described in Section 4.2.2 and 4.2.3, are provided as such plug-ins. Eclipse was used when developing the extensions that were necessary on WSMX in the form of adapters and an entry point.

4.2.2 Web Services Modeling Toolkit

The Web Services Modeling Toolkit (WSMT) is an Integrated Development Environment (IDE) for Semantic Web Services implemented in the Eclipse framework. Its functionality has already been described in Section 3.2.2.

WSMT was used when implementing the WSMO elements for the ontology, goal and Web service. Both the WSML text editor and the WSML visualiser were used. WSMT was also used to connect to the running WSMX instance to access the ontologies stored there, and to send and receive SOAP messages to test the implementation.

4.2.3 WSMO Studio

WSMO Studio is an open source Semantic Web Service and Semantic Business Process modeling environment for WSMO. It is offered as a set of plug-ins to Eclipse [22]. WSMO Studio offers functionality for ontology editing, WSMO elements editing, Semantic Business Process Modelling, WSML syntax validation, and a WSML editor.

WSMO Studio was used when problems were encountered with the WSML files written in WSMT. The problems were found and corrected when using the editor for WSMO elements. The editor generates the WSML based on inputs from the user in a form based view.

4.2.4 SoapUI

SoapUI is a tool for Web service testing [25]. The application is free and open source. SoapUI offers functionality for inspecting, invoking and developing Web services. In this project SoapUI was used to connect to the WSMX entry points, and to send and receive messages to and from these entry points.

4.3 Implementation Parts

In this section the different parts of the implementation will be explained. Details will be given on how each part was implemented and how it contributes to the final solution. The components that make up the implementation of the SWS scenario in Section 4.1 are the WSMO ontology descriptions for the goal, the ontology and the Web, the adapters that supports the communication with the entities outside WSMX, and the new entry point to WSMX. Each part is explained subsequently.

[25] http://www.soapui.org/

4.3.1 WSMO Ontology Descriptions

The WSMO ontology descriptions that were created to realise the SWS scenario is described in this section. The process of creating the different WSML files for each description is detailed together with any problems encountered. The WSML files can be seen in Appendix A, and are also included in the compiled WSMX distribution in the folder /resources/resourcemanager/masterthesis/(subfolders for each kind of description).

Ontology The ontology was created to describe the information sent in messages to and from the Barnes & Noble price search Web service. The name of the WSML for the ontology is *barnesNoblePrice-ontology.wsml*, and the WSML namespace is *http://www.ntnu.no/ontologies/bnpriceOntology*. The complete WSML file for the ontology is included in Appendix A.1.

A SOAP request and response sample from the Abundant Technologies Web page was used to determine the contents of the messages used to communicate with the Web service. The samples can be seen in Figure 9. From these samples it was determined that two ontology concepts were needed: *GetBNquote* and *GetBNQuoteResponse*. The two concepts have one attribute each; *sISBN* and *GetBNQuoteResult* respectively. This information is what the ontology describes.

The following is a sample SOAP request and response. The placeholders shown need to be replaced with actual values.

```
POST /WebServices/BNPrice/BNPrice.asmx HTTP/1.1
Host: www.abundanttech.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.abundanttech.com/webservices/BNPrice/GetBNQuote"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetBNQuote xmlns="http://www.abundanttech.com/webservices/BNPrice">
      <sISBN>string</sISBN>
    </GetBNQuote>
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetBNQuoteResponse xmlns="http://www.abundanttech.com/webservices/BNPrice">
      <GetBNQuoteResult>string</GetBNQuoteResult>
    </GetBNQuoteResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 9: Sample SOAP Request and Response for Barnes & Noble Web Service [12]

Web Service The Web service description was created to detail the Barns & Noble price search Web service, and how to interact with it. The name of the WSML file for the Web service is *WSBarnesNoble.wsml*, and the namespace is *http://www.ntnu.no/ontologies/WSBarnesNoble*. The complete WSML file is included in Appendix A.2.

The capability part of the Web service description details the pre and post conditions of the Web service. For this Web service the pre condition is that the input must be of the concept *GetBNQuote*. The post condition is that the output must be of the concept *GetBNQuoteResponse*. These concepts are both part of the Barnes & Noble ontology described above, and this ontology is thus imported in the capability.

The interface of the Barnes & Noble Web service description includes a choreography.

The choreography describes the interaction with the service from a client's perspective, and includes conditions on the input and output of the service. The choreography is divided into a state signature and a transition rule.

The state signature details the input and output, together with the appropriate access points to the service through a grounding. The state signature for this Web service can be seen in Figure 10. The input and output is described under the keywords **in** and **out**, where they are linked with the concepts of the Barnes & Noble ontology. The grounding of each element is a link to the WSDL service of the Abundant Technologies Barnes & Noble price search Web service. The value of the grounding is a WSDL service component identifier. The grounding is used when invoking the web service, as described in Section 4.3.2.

```
stateSignature bnStateSignature
     importsOntology
       _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

     in
          concept bnPrice#GetBNQuote withGrounding
          _"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#wsdl.interfaceMessageReference(bnprice/GetBNQuote/in0)"

     out
          concept bnPrice#GetBNQuoteResponse withGrounding
          _"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#wsdl.interfaceMessageReference(bnprice/GetBNQuoteResponse/out0)"
```

Figure 10: The State Signature of the Barnes & Noble Web Service Description

The transition rule for the Web service details the conditions that apply between the input and the output of the service. In this case it states that for each input of the concept *GetBNQuote*, an instance of the concept *GetBNQuoteResponse* is added containing the result from the Web service invocation.

Goal The goal was created to describe the information desired by the user. The goal description is very similar to that of Web service. The name of the WSML for the goal is *BarnesNobleGoal.wsml*, and the WSML namespace is *http://www.ntnu.no/ontologies/BarnesNobleGoal*. The complete WSML file for the goal is included in Appendix A.3.

The goal has a capability with pre and post conditions which are the same as the ones for the Web service description. The interface is also the same with a choreography with a state signature and a transition rule.

When a user wants to achieve this goal, he sends in the goal with input data. This input data is identified by the keyword **instance**. Such a goal was created for testing purposes, and has the name *BarnesNobleGoalWithInstances.wsml*. The name space for the goal with an instance is *http://www.ntnu.no/ontologies/BarnesNobleGoalWithInstances*. This goal also defines the ontology it uses in an import. The ontology importing and the instance definition can be seen in Figure 11. The rest of the file is the same as for BarnesNobleGoal.

4.3.2 WSMX Adapters

When WSMX communicates with service providers, the information is communicated by sending and receiving SOAP messages, with contents described in XML. The possible

```
ontology _"http://www.ntnu.no/ontologies/getBNQuoteWithInstances"
    importsOntology _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

instance
    testBNGoal memberOf bnPrice#GetBNQuote
    bnPrice#sISBN hasValue "0321155556"
```

Figure 11: The Ontology and Instance Parts of the Goal with Instance

contents of these XMLs are given in the WSDL of the service. Since WSMX itself uses WSML for internal representation of information, this information has to be translated into and from XML sent and received from the services. WSMX uses a lowering adapter to translate from WSML to XML, and a lifting adapter to translate from XML to WSML. Such adapters have been written for the Barnes & Noble Web service based on the WSDL of the service. The code for the adapters can be seen in Appendix B.1. The adapters are described below.

Lowering Adapter The method *getXML* in *BarnesNobleAdapter.java* is used to translate from the internal WSML representation to a XML that can be sent to the Web service. The method is called by the invoker during synchronized invocation though the method *syncInvoke*, which again uses the method *inlineWSMLtoXMLAdapter*. In this method the right adapter is chosen based on the non-functional property **http://owner** in the Web service description. The code added in the class *Invoker.java* to accomplish this can be seen below.

```
// Master thesis book search
if (theOwner.contains("http://BarnesNoble")) {
        // Do BarnesNoble adaption here
        try {
                BarnesNobleAdapter adapter = new BarnesNobleAdapter();
                logger.debug("Translating BarnesNoble instance to XML: ");
                org.w3c.dom.Document doc = adapter.getXML((Instance) data
                                .get(0));
                soapAction = "http://www.abundanttech.com/webservices"
                +"/BNPrice/GetBNQuote";

                return new Pair(null, doc);
        } catch (Exception e) {
                e.printStackTrace();
                return new Pair(null, null);
        }
}

// THE TP SHOWCASE
else if (theOwner.contains(TP_Showcase_Constants.tpscOwnerPath)) {
.....
```

The string *soapAction* is used to set the value for the soap action element in the XML. This element must contain the URL of the service you wish to invoke, since the Abundant Technologies Web service is written in .NET. The original code in the *syncInvoke* method only added the name of the service and not the entire URL. The code was changed to achieve this desired functionality.

The method *getXML* takes as input the instance describing the input for the Web service. If the instance is of the concept *getBNQuote* from the *bnpriceOntology*, the value of the *sISBN* attribute is retrieved. A runtime exception is cast if the instance is not of this concept. A string is used to build an XML in a form that is accepted by the Barnes & Noble Web service for the **getBNQuote** operation. The value of the *sISBN* attribute is included in this XML. The XML in the string is then parsed and serialized before the method returns it as a **org.w3c.dom.Document** object.

Lifting Adapter The method *getWSML* in *BarnesNobleAdapter.java* is used to translate the XML response from the Web service to an internal WSML representation. The method is called by the invoker during synchronized invocation though the method *syncInvoke*, which again uses the method *inlineXMLtoWSMLAdapter*. In this method the right adapter is chosen in the same way as when translating from WSML to XML.

The *getXML* method takes as input a string with the XML response from the Web service, and the endpoint grounding (this object is not used). The method uses an XSL [26] file to perform a XSL-Transformation of the XML string. The result of the transformation is a WSML file representing the response from the Web service. The XSL file can be seen in Appendix C. An XML translator takes the XML string and the XSL (represented as a **FileInputStream** object) as input to perform the translation to WSML. The translated string is added as the content of a **WSMLDocument** object, which is returned by the method.

4.3.3 WSMX Entry Point

As described in Section 3.2, the WSMX server offers a set of entry points that can be used to access its functionality. One of these entry points is *achieveGoal*, which can be used to send in a WSML Goal description. This entry point could be used for our scenario, but it is more convenient for anyone requesting the service the book price search offers to send and receive information represented in XML. A new entry point to the WSMX server has been created to achieve this.

The new entry point is called **BookPriceSearch**, and provides the operation *getBookPrice* with an ISBN as input. The result is returned in a *getBookPriceResponse*. The WSDL for the entry point is available at
http://localhost:8050/axis/services/BookPriceSearch?wsdl. The functionality of the operation has been implemented in *BookPriceSearch.java* (See Appendix B.2 for source code).

The method *getBookPrice* is executed each time a new request is sent to the entry point operation. The method takes a string with an ISBN as input, and returns a price for the book (or an error if the price is not found). In the method a string with the contents of a WSML goal description with an instance is created. This description is exactly the same as for *BarnesNobleGoalWithInstances.wsml* in Section 4.3.1. The ISBN in the input is added as the value of the *sISBN* attribute of the *GetBNQuote* concept. This goal is then invoked by using the MBean server to send a *achieveGoalFullResponse*

[26] XSL - Extensible Stylesheet Language

request to the communication manager, with the goal as input. The instance created from the invocation is retrieved, and the value of the *GetBNQuoteResult* attribute of the *GetBNQuoteResponse* concept is returned to the requester in a *getBookPriceResponse*.

The new entry point is created when the project is built with the Ant [27] build files provided in the WSMX project. A few lines had to be added to the build file *build.xml* in the webservices folder:

```
<delete dir=
"\${basedir}/webapps/webservices_temp/BookPriceSearch/ie/deri/webservices/"
/>

<copy file=
"\${basedir}/build/classes/ie/deri/webservices/BookPriceSearch.class"
todir=
"\${basedir}/webapps/webservices_temp/BookPriceSearch/ie/deri/webservices"
>
</copy>

<jar destfile=
"\${basedir}/webapps/webservices_temp/services/BookPriceSearch.aar"
basedir=
"\${basedir}/webapps/webservices_temp/BookPriceSearch/"
/>
```

The first line deletes the old webservices_temp folder. The second line copies the Class file for the entry point to the webservices_temp folder, and the last line compiles the entry point's JAR [28]. In addition to these lines a XML describing the service was created. This XML is used during the build process, and can be seen in Appendix D.

4.4 Final Solution

After the development process was finished, the Semantic Web Service worked according to the scenario in Section 4.1. By sending a request XML with an ISBN, the response is an XML with the price ffor the book. The solution consists of a SWS web service defined by the ontology, goal and Web service descriptions. An entry point receives the XML from the requester, and invokes the SWS by providing a goal with an instance created with input from the XML. The SWS then invokes the appropriate provider Web service to fetch the book price. The result from this invocation is returned to requester through a response XML. The lowering and lifting adapters that were created, is used during the Web service invocation. An overview of the final solution can be seen in Figure 12. An explanation on how to run WSMX and use the SWS can be found in Appendix E, together with a description of the contents of the provided attachments.

[27] Apache Ant - a Java-based build tool
[28] JAR - Java ARchive: Distribution of Java classes and associated metadata

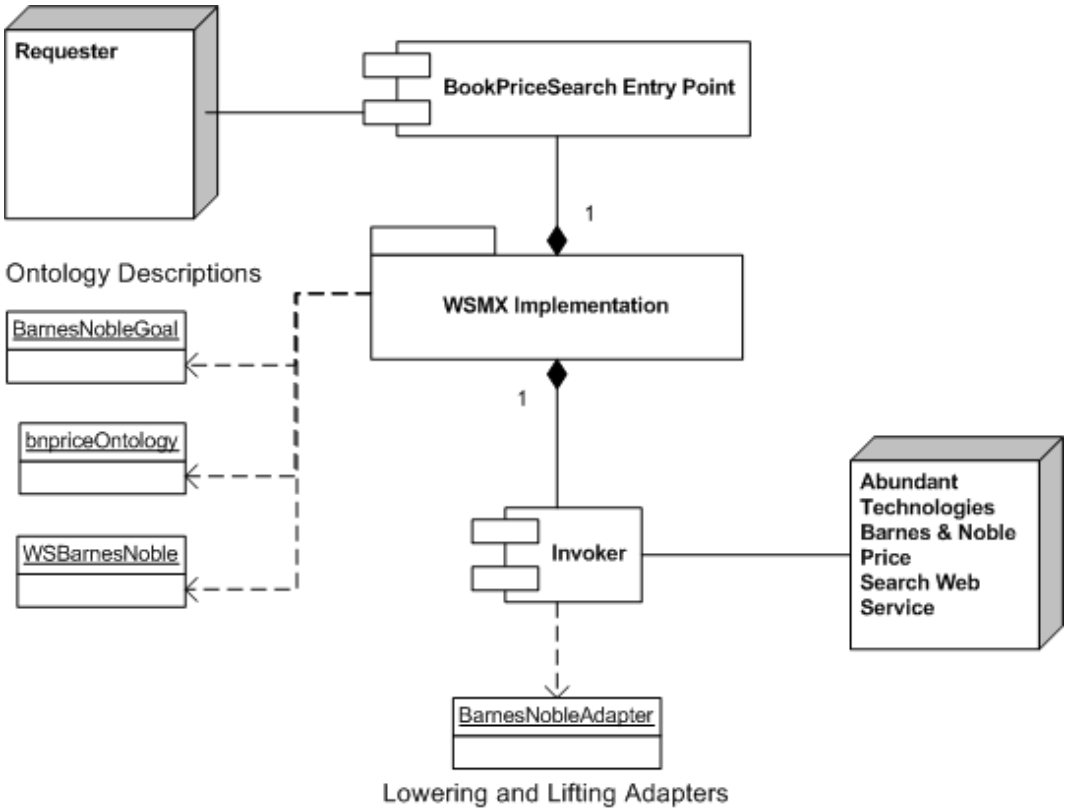

Figure 12: Overview of the Final Soulution

5 Evaluation

In this section the current implementation of the WSMO framework will be evaluated. The evaluation is performed on the basis of experiences gathered during the implementation of the SWS scenario described in Section 4. The evaluation is divided into three main areas: the WSMX implementation, documentation, and the available tools. Each area is evaluated in the subsequent sections.

5.1 WSMX implementation

WSMX offers functionality for running Semantic Web Services based on WSMO ontology descriptions. Its implementation is still under development, and because of this, some parts of the system is more complete than others. The evaluation in this section will discuss some of the problems that are not yet solved, in addition to the experiences gathered during the implementation of the SWS scenario in Section 4. After the development was finished the scenario worked as intended, which shows that WSMO has great potential for providing fully working Semantic Web Services.

Table 2 shows an overview of the different parts of the implementation, the number of code lines for each part, and the time spent on it. For the ontology descriptions the term code line is chosen to mean each time a keyword is used, like for instance **wsmlVariant** which is the first keyword of every WSML file. If there are multiple keywords on one line, the line is only counted once. Ending keywords, like **endNonFunctionalProperties**, are not counted. For the Java code a code line is any line ending with a semicolon. The time spent on each part is approximate, and includes times spent on writing code, reading documentation, and testing.

Table 2: Overview of Complexity for each Part of the Implementation

Part	**No. of Lines**	**Time Spent**
Lowering Adapter	45	2 work days
Lifting Adapter	14	6 hours
Entry Point	17	4 work days
Ontology	23	1 work day
Goal	35	2 work days
Web Service	37	2 work days
Goal w/Instance	40	2 hours

5.1.1 WSMX Components

In the WSMX Documentation[19] a summary of the status for each component comprising the system is given. This information is represented in Table 3. As can be seen, most of the components are working and stable, but some work still needs to be done on some of the components before they can fulfill their goals for functionality.

During the implementation of the SWS scenario the problem mentioned in the WSMX Documentation that was most visible, was the need to manually make code changes and additions to the Invoker component. A lot of time was used to understand how to

Table 3: Status for the Components of WSMX

Component	**Current Status**
Core	Stable, but might be changed if new components or functionality is introduced
Choreography	Stable
Communication Manager	Stable
Data Mediator	"'Related-by"' mapings are not implemented
Invoker	Generic grounding is being developed to remove the need to change the code manually. Complete support for REST-based Web services will be added
Orchestration	Only used as stand-alone for tests. At the moment the Choreography component covers for some of the functionality of this component in the WSMX execution semantics
Parser	Used in the WSMX execution semantics. Other components uses WSMO4J directly
Resource Manager	No persitency support (DB Storage), only in-memory representation by using the file system
Service Discovery	Relatively stable. The QoS Discovery currently a part of this component should be moved to a separate component
Web Service Discovery	Discovery of services based on keywords (Keyword-Based) and the postcondition part of the service capability (Lightweight) is currently supported. A more advanced discovery method called Lightwight DL will be implemented

create the lowering and lifting adapters that was needed to translate from WSML to XML and vice a versa. The examples included in the WSMX repository was of great help, but they are quite large, and a lot of the code was not applicable for the SWS scenario. This meant that a lot of time had to be used to separate the pieces of code that were equal for any adapter, and the ones that were specially made for the adapters in each example. More time was spent on the lowering adapter than the lifting adapter, as can be seen in Table 2. The code for the lifting adapter is less complex because the actual lifting is carried out by XSL-Transformation. The XSL used was quite easy to create, as an XSL from an example could be used in a straight forward way.

The setting of a SOAPAction for the SOAP-message was another problem that was encountered with the invoker. The Web service from Abundant Technologies that was used in the scenario was implemented in .NET, which meant that the SOAPAction had to be set to a full URL to the operation the message was supposed to invoke. The original code that set this SOAPAction only set it as the name of the operation. The code was altered to set it with a full URL.

Four work days were used on the entry point to the WSMX for the book price search, which is a lot of time for only 17 lines of code. There were a few examples of entry points with the code, but these had been commented out, and did not work after being uncommented. The problem was that this did not have to mean that the code was not

working. The errors could come from another part of the example, like for example the ontology descriptions or the adapters. Because of this it was not possible to determine if the code could be replicated in the new entry point. In addition to this it was again a problem that the example was much more complex, so the code was difficult to understand in order to single out the important parts. After a lot of trial and error with elements of the example code the entry point was constructed to call an operation through the Communication Manager, and use the result to return a response to the requester.

By these examples of problems that were encountered during the creation of the lowering and lifting adapters and the entry point, it is easy to see that these processes should be automated. The need to make code changes to set up a Semantic Web Service is not a good solution. The creation of a Semantic Web Service should only include a configuration of the server, and to provide the belonging ontology descriptions. The problem now is that the person setting up the service needs knowledge about specific parts of the implementation, which makes WSMX cumbersome to use. If Semantic Web Services is to gain acceptance in the industry, an ease of use is of uttermost importance.

In the current version of WSMX the Resource Manager uses WSML files stored in the local file system. To be able to locate these files, the path to each file that is to be read during start up of the server has to be set in a configuration file (config.properties). This is of course a very cumbersome way of adding new ontology descriptions to the server. By using a database, as mentioned in the WSMX documentation, this could be solved much easier. This would also add support of persistency which is also a functionality that is needed, but not supported by the current implementation. The introduction of a data base would improve the ease of use for WSMX.

5.1.2 Ontology Descriptions

The ontology descriptions for the SWS scenario were created with help from the included examples, and by reading the WSML and WSMX documentations (see Section 5.2.2). A few problems were encountered during the creation of the descriptions, and most of these came from trying to replicate the structure of the descriptions in the examples. The problems centered on references to the elements of the ontology from the goal and Web service descriptions.

While creating the pre and post conditions for the goal and Web service, a problem was encountered where the goal was not matched with the Web service. The reason for this was that in the conditions, the keyword **concept** had to be placed before the reference to the concept in the ontology. The keyword is not used in the examples, and I have not been able to figure out why it is not needed there. The matching of goal and Web service worked fine after adding the keyword. Because the example worked fine without the keyword, it took some time before realizing that this was the problem.

During the creation of the choreography for the goal and Web service, a problem was encountered where the two choreographies were not able to cooperate during invocation. The problem was again the missing **concept** keyword in front the ontology concept reference. This key word was not used in the examples here either. The solution to the problem was to use WSMO Studio to create the choreography. In WSMO Studio the

choreography could be set up in an editor by fetching the right references straight from the ontology. The code for the description was then generated automatically. There might actually have been other problems that also were corrected after using WSMO Studio to auto generate the choreography part of the description.

There were two reasons for the troubles that were encountered during the creation of the ontology descriptions. The first were some unexplained differences between what works in the examples and what worked in the scenario. If the examples were changed to follow the standard for ontology references, this problem would be solved, as the examples would be easier to follow. The other reason was the process of writing the descriptions. This is discussed further in Section 5.3.

5.1.3 Testing Posibilities

Testing is an important part of a development process, and in this case the testing was performed in unit tests. In the WSMX repository there are three unit tests that were used to test the Semantic Web Service during development:

- **AchieveGoalExecutionSemanticsTest -** was used to test the SWS by simulating invocation of the achieveGoal entry point. This unit test simulates both the discovery of the Web service and its invocation.
- **DiscoveryOnlyExecutionSemanticsTest -** was used to test the dicovery of the Web service, with keyword-based and lightweight discovery.
- **InvokeOnlyExecutionSemanticsTest -** was used to test the invocation of the Web service, including the use of the lifting and lowering adapters.

These unit tests were of great help when debugging the ontology descriptions and the adapters. By using the debug feature in Eclipse, the whole process of discovery and invocation could be monitored by following the setting of each variable and object. While running these tests, a lot of comments were written to the output console describing each step in the process. These logging outputs are part of the framework and helped locating any problems that were encountered.

No pre-made unit tests were available while working on the new entry point to WSMX. To be able to debug the entry point, outputs had to be created programmatically. These outputs could be seen in the WSMX console while running the server. This meant that the project had to be built anew for each time a change was made to the entry point. The build process took about 1.5 minute. This was quite cumbersome, and it was also a drawback that the debug feature of Eclipse could not be used. This was one of the reasons for the long development time for the entry point.

5.2 Documentation

While implementing the SWS scenario, important documentation for WSMX was used for aid and guidance. This documentation is evaluated in this section. The evaluation

deals with the documentation of source code, the WSMX documentation [19], and the WSML documentation.

5.2.1 Code Documentation

An understanding of the source code is essential for anyone who wants to create a Semantic Web Service in the current implementation of WSMX. The source code needs to be altered for the lowering and lifting adapters, and to create a new entry point. A thorough documentation of the code is important in this respect, since it makes it easier to understand what the code accomplishes. Unfortunately the documentation of the source code for WSMX is quite insufficient. Most classes and methods have some kind of documentation, but it is very limited and is usually a very short description of what the class or method accomplishes, not how this is achieved. A sample of a typical source code comment for the WSMX implementation can be seen below:

```
@Exposed(description = "Make a synchronous invocation of the Web service.")
   public List<Entity> syncInvoke(WebService sws, List<Entity> data,
         EndpointGrounding grounding, Ontology ontology)
         throws ComponentException {
      try {
         WSDL1_1EndpointGrounding theGrounding =
               (WSDL1_1EndpointGrounding) grounding;
         Pair<SOAPElement, Document> soapParts = inlineWSMLtoXMLAdapter(sws,
               grounding, ontology, data);
         ....
```

This kind of comment is not of much help while trying to understand the code. By providing a detailed method comment, anyone working with the WSMX implementation would have a better basis for understanding how it works.

The biggest problem is the almost total lack of documentation of the code for the examples that are included in the WSMX repository. The examples are the only source of similar code when creating a new SWS, and when the code is not commented it gets difficult to understand. This resulted in a lot of trial and error during the implementation of the SWS scenario, before the parts of the examples that were applicable for this specific scenario were found. Better commenting of the examples could have decreased the time spent on the adapters and the entry point drastically.

5.2.2 Publications

A few publications were used during the implementation. The most important was the WSMX documentation [19], where several important aspects about WSMX are described together with an implementation guide. Documentation for WSML was also used during the implementation. The publications are evaluated below.

WSMX Documentation The WSMX Documentation explains the most important aspects of WSMX. It starts off with a short explanation of the architecture that is easy to understand, and gives an introduction to the system. The next part describes

how to install and use WSMX. The installation section includes a description on how to download and build WSMX from the SVN repository. The next section explains how to communicate with the server, and how to monitor it. A short introduction is also given to WSMT and WSMO Studio. This part of the documentation is easy to understand, and gives all the information needed to install and use WSMX.

The third part of the documentation explains the process of developing and running Semantic Web Services. It also gives a short introduction to unit testing in WSMX, and a short introduction to the available examples. This is the most important part of the documentation in regard to the development process executed in this project. The section starts with an overview of the elements for a development process for a SWS on WSMX. It explains how to create an ontology based on a WSDL by looking at the messages that is being sent and received, but the creation of a web service and a goal is not detailed. The creation of adapters is not explained in detail either, but the reader is referred to the examples and certain methods in the Invoker. Later in the section a short description is given on how to create an interface with a choreography and transition rules for the web service and goal descriptions.

The explanations in this part of the documentation are unfortunately a bit too thin to offer enough help for anyone that is new to the creation of SWS for WSMX. The parts explaining the creation of Web service and goal descriptions should contain a thorough description on how to create these, or at least include a reference to a WSML documentation where the information can be found. The descriptions of the adapters could also be extended, but a full guide on how to create these would be unnecessary. It is more important that the example code gets better commented as mentioned above.

The fourth part of the documentation provides a simple but good overview of the different components of WSMX. Each components function/service and motivation is stated, and the current status is provided. The overview is a nice reference when trying to understand how the system works and what tasks each component have. A short FAQ [29] is included in the Appendix. None of the problems encountered during implementation were answered here, since it focus more on the main development of the WSMX system. It would be a good idea to either extend this FAQ, or create a new one, to provide answers to SWS implementation questions. This could be of great help to any new users.

WSML Documentation The documentation for WSML was not much used during the implementation, but parts of the WSML description [6] and a document explaining WSML choreographies and transition rules [13] were used to some extent. Both documents provided answers to the questions that needed answering at the time of their use. The use of examples in both documents was of great help to understand the concepts that were investigated.

[29] FAQ - Frequently Asked Questions

5.3 Tools

During the implementation of the SWS scenario, a set of tools were used as aid in the development process. Two of these are specially made for WSMO, namely WSMT and WSMO Studio. These tools are the subject to a short evaluation in this section. The other tools that were used are more general, and are not within the scope of an evaluation of the WSMO framework.

The Web Service Modeling Toolkit was used from the start of the development process. It was used to create the WSML files for the ontology descriptions of the ontology, the goal and the Web service. The creation of the ontology went smoothly by using the WSML Visualizer. The text editor was used to create the other ontology descriptions. While editing the files in this editor WSMT provided validation that gave feedback on any syntactical errors and other structural errors. This was a help when not being familiar with WSML.

When problems were encountered with the references to the elements of the ontology from the goal and Web service descriptions (as described in Section 5.1.2), WSMT did not discover this error. The errors were found during unit testing of WSMX. Because WSMT did not locate these errors, WSMO Studio was tried in hope of finding and solving them. In WSMO Studio the user can use a WSMO Editor to edit the ontology descriptions. This editor automates the process of creating ontology descriptions in WSML, and offers functionality where the user chooses ontology references straight from the actual ontology. By using this editor the problems were solved. The editor is very intuitive and makes the creation of ontology descriptions easier than writing them in a text editor, especially for someone not familiar with WSML. WSMT does not have such an editor, so by adding this feature WSMT could be made better.

6 Discussion and Conclusions

In this section conclusions are made from the evaluation of the WSMO framework implementation. Other work related to this project is described, and the threats to the validity of the evaluation are pointed out. The section is concluded with suggestions to further work.

6.1 Related Work

No other evaluations of the WSMO framework implementation have been found, but the Semantic Web Service Challenge is working on a book where an evaluation will be performed. The goal of the SWS challenge is to develop a common understanding of various technologies intended to facilitate the automation of mediation, choreography and discovery for Web Services using semantic annotations [16]. They are doing this by implementing a series of scenarios on different framework implementations. WSMO is one of these. The book has not yet been released, but is going to be published at Springer in the near future (according to the SWS Challenge Web page, the book should be sent to Springer on June 1. this year).

An evaluation is performed in [11] that considers the WSMO Studio tool. The evaluation only focuses on aspects of WSMO Studio, and is thus only partly related to this project because the main focus of the evaluation performed here is the WSMO framework implementation, not WSMO Studio. The same applies for the short evaluation of WSMT in [20].

6.2 Threats to Validity

Some elements of how this project has been carried out may lead a reduced validity of the results from the evaluation. The evaluation in this master thesis was carried out by only one person, and one could argue that an involvement of different persons in the evaluation of the WSMO framework implementation would have given more accurate results. By letting a group of people with different backgrounds implement the same scenario on WSMX, the results would probably be more accurate and the different persons would probably encounter different problems. Unfortunately an involvement of for instance four persons in an evaluation of WSMO is beyond the scope of this master thesis. That being said, my background as a student with a good bit of knowledge about the area of computer science, but not that much knowledge about Semantic Web Services, could be a typical future user of WSMO/WSMX. The field of SWS is young, and not great many people have knowledge about it. Because of this, a large group of SWS developers in the future would probably be close to my background. There is reason to believe that this group will run into the same problems encountered in this project, and that removing these will improve the quality of the framework implementation.

The scenario that has been implemented to aid the evaluation is not the most advanced that could have been invented. By making the scenario more complex, all aspects of WSMX could have been tested. This is unfortunately beyond the scope of this

project. The scenario that has been implemented covers the most basic and important elements of WSMX, and the results gathered from the evaluation is thus of value. An implementation of different scenarios would also lead to a better basis for evaluating the framework implementation, but is also beyond the scope of this project.

By implementing the scenario (or several scenarios) on different framework implementations, other results might have been found. Another possibility would be to develop a separate framework to be better able to reason about the WSMX implementation, and how it works. Both these suggestions are beyond the scope of this master thesis. All in all I believe that the results from this project are valid under the given circumstances, even though the results from the evaluation could be improved and extended by making some alternations.

6.3 Conclusions

In this project we have looked at the WSMX implementation of the WSMO framework. This framework was chosen for evaluation because it looks to be one of the most promising to fulfill the goals of Semantic Web Services, and because it has a working implementation with available source code. A scenario was defined and implemented for WSMX in order to get hands on experience from the WSMO framework implementation. The evaluation was based on this implementation and included an evaluation of the WSMX implementation itself, and also the available documentation and tools.

The evaluation showed that some improvements still need to be made to WSMX, but that it is currently able to fulfill the goals of Semantic Web Services that were tested. The SWS scenario implemented in this master thesis worked as planned at the end of the development, which shows that the WSMX implementation of WSMO has great potential for providing fully working Semantic Web Services. While working on the scenario some flaws with WSMX were found, flaws that are mostly an inconvenience to the one developing the services. Some missing functionality of WSMX was also found by investigating the WSMX documentation. This functionality included a part of the mediation component, lack of database storage, and more advanced Web service discovery.

The current implementation of WSMX has two major shortcomings from a SWS developer's point of view. The first is that the developer currently has to make code changes to implement a service. This makes the development cumbersome, and a solution should be found where these changes are handled automatically. This is currently being worked on by the WSMX developers. With the current solution, the documentation of the source code is not well enough commented. This makes the development process even more difficult as it is today. The second shortcoming from a developer's point of view, is that the examples that are currently included with the implementation are difficult to understand, and needs better documentation. This lack of documentation includes both source code comments and general descriptions. It is important to provide good examples since this can be of vital help for new SWS developers.

Two other important elements that are needed to increase the ease of developing

Semantic Web services are good documentation and adequate tools. The WSMX documentation is inadequate when it comes to explaining the creation of ontology descriptions, and the code changes that is currently needed. The documentation of installation and use of WSMX is satisfactory. Both the WSMT and WSMO Studio tools were used in the development in this project. Both tools helped in the creation of ontology descriptions, and probably decreased the time spent on this part of the implementation.

Even though there is room for some improvements of the WSMX implementation of the WSMO framework, the work in this master thesis has shown that things are moving in the right direction. By making the changes and additions that has been suggested here, the product could be made even better.

6.4 Further Work

This master thesis has evaluated the WSMX implementation of the WSMO framework on the basis of the development of a simple SWS scenario. This SWS made use of the most basic elements of WSMX like discovering and invoking services, but did not include for instance mediation. By implementing a bigger and more advanced scenario the results of the evaluation would probably have produced more and different results, and other suggestions for improvements. It would therefore be interesting to carry out an evaluation with a more complex SWS scenario.

By comparing two or more framework implementations, an evaluation could probably come up with even more advanced findings and results. Such an evaluation could also include the implementation of a new SWS framework, to see of things could be done in a better way than in the frameworks that are available today.

References

[1] Meteor-s: Semantic web services and processes (web page). http://lsdis.cs.uga.edu/projects/meteor-s/, 22.10.2007.

[2] Web information service modeling (wisemod). http://www.idi.ntnu.no/ guttors/-wisemod/.

[3] Web services description language for java toolkit. http://sourceforge.net/projects/wsdl4j.

[4] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, , and K. Verma. Web service semantics - wsdl-s. technical note, university of georgia and ibm. http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.pdf., 12.11.2007, apr. 2005.

[5] Tim Berners-Lee, J. Handler, and O. Lassila. The semantic web. *Scientific American, 284(5)*, 2001.

[6] J. De Bruijn, H. Laursen, R. Krummenacher, L. Predoiu, M. Kifer, A. Polleres, and D. Fensel. D16.1v0.21 the web service modeling language wsml. http://www.wsmo.org/TR/d16/d16.1/v0.21/, 28.3.2008, 2005.

[7] J. De Bruijn, H. Laursen, A. Polleres, and D. Fensel. The web service modeling language wsml: An overview. http://www.wsmo.org/wsml/wsml-syntax, 28.3.2008, 2005.

[8] Cabral, Domingue, Motta, Payne, and Hakimpour. Approaches to sematic web services: an overview and comparisons. 2004.

[9] L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, and B. Norton. Irs-iii: A broker for semantic web services based applications. ISWC 2006, LNCS 4273, pp. 201Ű214, 2006.

[10] DAML. Owl-s web page. http://www.daml.org/services/owl-s/, 29.9.2007.

[11] M. Dimitrov, A. Simov, V. Momtchev, and Konstantinov M. Wsmo studio Ű a semantic web services modelling environment for ws. http://www.wsmostudio.org/doc/WSMO-Studio-ESWC2007.pdf, 28.05.2008.

[12] Abundant Technologies IT Consulting Experts. Abundant technologies web page. http://www.abundanttech.com/index.htm, 16.04.2008, 2002-2004.

[13] D. Fensel, A. Polleres, and J. de Bruijn. D14v0.4. ontology-based choreography. http://www.wsmo.org/TR/d14/v0.4/, 16.05.2008, 2007.

[14] K. Fujii and T. Suda. Dynamic service composition using semantic information. Proceedings of the 2nd International Conference on Service Oriented Computing (ISOC04), 2004.

[15] Miniwatts Marketing Group. Internet usage statistics. http://www.internetworldstats.com/stats.htm, 12.12.2007.

[16] Stanford Logic Group. Semantic web services challenge wiki. http://www.sws-challenge.org, 28.05.2008, 2008.

[17] A. Haller, E. Cimpian, A Mocan, E Oren, and C. Bussler. Wsmx - a semantic service-oriented architecture. Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference, p321- 328 vol.1 , http://www.wsmx.org 11.10.2007.

[18] Hao He. What is service-oriented architecture. http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html, 30.9.2007, 2003.

[19] M. Herold. Wsmx documentation. http://www.wsmx.org:8080/wsmxsite/papers/documentation/WSMXDocumentation.pdf, 03.05.2008, 2008.

[20] M. Kerrigan, A. Mocan, M. Tanler, and D. Fensel. The web service modeling toolkit - an integrated development environment for semantic web services. The Semantic Web: Research and Applications, p789-798, 2007.

[21] KMI. Internet reasoning service. http://kmi.open.ac.uk/projects/irs/.

[22] Ontotext Lab. Wsmo studio web page. http://www.wsmostudio.org/, 18.05.2008, 2005-2008.

[23] T. B. Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. 1999.

[24] B. Leiner, V. Cerf, D. Clark, R. Kahn, D. Kleinrock, D. Lynch, J. Postel, L. Roberts, and S. Wolff. A brief history of the internet. http://www.packet.cc/history-files/Brief-History.html, 10.12.2007, 2000.

[25] D. Martin, M. Burnstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s: Semantic markup for web services. Member Submission, World Wide Web Consortium (W3C), nov. 2004.

[26] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. IEEE Intelligent Systems, Special Issue on the Semantic Web, 16(2):46Ű53, 2001.

[27] D. Roman, U. Keller, H. Laursen, J. Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. Applied Ontology 1, p77-106, http://www.wsmo.org 22.10.2007, 2005.

[28] Ø. Skytøen. Semantic web services: an evaluation of existing frameworks, 2007.

[29] I. Toma and N. Steinmetz. D16.1v0.3 wsml language reference. http://www.wsmo.org/TR/d16/d16.1/v0.3/, 18.5.2008, 2008.

[30] K. Verma, K. Gomadam, A. P. Sheth, J. A. Miller, and Z. Wu. The meteor-s approach for configuring and executing dynamic web processes. Technical report, June 2005, http://lsdis.cs.uga.edu/projects/meteor-s/techRep6-24-05.pdf, 21.10.2007.

[31] K. Verma, R. Mulye, A. P. Sheth, J. A. Miller, and K. Gomadam. A semantic template based designer for web processes. Proc. of the Third International Conference on Web Services (ICWS, 2005).

[32] R. K. Yin. *Case Study Research Design and Methods, Third Edition.* Sage Publications, 2003.

Appendix

A WSMO Ontology Descriptions

In this section the ontology descriptions for the ontology, goal and Web service is included.

A.1 Barnes & Noble Ontology

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.ntnu.no/ontologies/bnpriceOntology#",
   dc _"http://purl.org/dc/elements/1.1/",
  xsd _"http://www.w3.org/2001/XMLSchema#" }

ontology barnesNoblePriceOntology
   nonFunctionalProperties
   dc#description hasValue "Simple Ontology describing a
      Barnes & Noble ISBN price search"
    dc#subject hasValue "Book Price Search"
    dc#identifier hasValue _"http://www.ntnu.no/ontologies/bookOntology"
    dc#title hasValue "Ontology for a Book Price Search"
    dc#format hasValue "text/html"
    dc#language hasValue "en-US"
    dc#creator hasValue "Oyvind Skytoen"
    dc#date hasValue "2008-05-02"
   endNonFunctionalProperties

concept GetBNQuote
   nonFunctionalProperties
   dc#description hasValue "Request for the price of a book"
  endNonFunctionalProperties
  sISBN impliesType  (1 1) _string
   nonFunctionalProperties
      dc#description hasValue "The ISBN number of a book"
    endNonFunctionalProperties

concept GetBNQuoteResponse
   nonFunctionalProperties
   dc#description hasValue "Response from a price search"
  endNonFunctionalProperties
  GetBNQuoteResult impliesType _string
   nonFunctionalProperties
      dc#description hasValue "The price of the book"
    endNonFunctionalProperties
```

A.2 Barnes & Noble Web Service

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.ntnu.no/ontologies/BarnesNobleWS#",
   dc _"http://purl.org/dc/elements/1.1/",
  xsd _"http://www.w3.org/2001/XMLSchema#",
  bnPrice _"http://www.ntnu.no/ontologies/bnpriceOntology#",
  wsml _"http://www.wsmo.org/2004/wsml#",
```

```
  wsmostudio _"http://www.wsmostudio.org#" }


webService _"http://www.ntnu.no/ontologies/BarnesNobleWS"
   nonFunctionalProperties
   wsmostudio#version hasValue "0.7.3"
    wsml#endpointDescription hasValue
      _"http://www.abundanttech.com/webservices/BNPrice/#wsdl.service(BNPrice)"
    dc#title hasValue "Barnes & Noble Price Search WS"
    _"http://owner" hasValue _"http://BarnesNoble"
    dc#format hasValue "text/html"
    dc#language hasValue "en-US"
    dc#creator hasValue "Oyvind Skytoen"
    dc#date hasValue "2008-05-02"
  endNonFunctionalProperties

   importsOntology
   _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

capability _"http://www.ntnu.no/ontologies/BarnesNobleWS#capability"

   importsOntology
   _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

   sharedVariables ?request

   precondition bnPricePrecon
   nonFunctionalProperties
      dc#description hasValue "The input has to be a getBNQuote"
    endNonFunctionalProperties
    definedBy
      ?request memberOf bnPrice#GetBNQuote.

   postcondition bnPricePostcon
   nonFunctionalProperties
      dc#description hasValue "The output is a GetBNQuoteResult"
    endNonFunctionalProperties
    definedBy
      ?response memberOf bnPrice#GetBNQuoteResponse.

interface _"http://www.ntnu.no/ontologies/BarnesNobleWS#interface"

   importsOntology
   _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

  choreography barnesNobleChoreography
   stateSignature bnStateSignature
      importsOntology
         _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

         in
            concept bnPrice#GetBNQuote withGrounding
            _"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl
         #wsdl.interfaceMessageReference(bnprice/GetBNQuote/in0)"

      out
         concept bnPrice#GetBNQuoteResponse withGrounding
            _"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#
```

```
        wsdl.interfaceMessageReference(bnprice/GetBNQuoteResponse/out0)"

    transitionRules _"http://www.ntnu.no/ontologies/BarnesNobleWS#transitionRules"
        forall {?request} with
        (?request memberOf bnPrice#GetBNQuote
       ) do
        add(_#1 memberOf bnPrice#GetBNQuoteResponse)
     endForall
```

A.3 Barnes & Noble Goal

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.ntnu.no/ontologies/BarnesNobleGoal#",
   dc _"http://purl.org/dc/elements/1.1/",
  xsd _"http://www.w3.org/2001/XMLSchema#",
  bnPrice _"http://www.ntnu.no/ontologies/bnpriceOntology#",
  wsmostudio _"http://www.wsmostudio.org#" }

goal _"http://www.ntnu.no/ontologies/BarnesNobleGoal"
  nonFunctionalProperties
   dc#type hasValue _"http://www.wsmo.org/2004/d2#goals"
    wsmostudio#version hasValue "0.7.3"
    dc#title hasValue "Barnes & Noble price search goal"
    dc#format hasValue "text/plain"
    dc#language hasValue "en-US"
  endNonFunctionalProperties

  importsOntology
   _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

capability _"http://www.ntnu.no/goals/BarnesNobleGoal#capability"
  importsOntology
   _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"

   sharedVariables ?request

   precondition bnPricePrecon
   nonFunctionalProperties
      dc#description hasValue "The input has to be a getBNQuote"
    endNonFunctionalProperties
    definedBy
      ?request memberOf bnPrice#GetBNQuote.

   postcondition bnPricePostcon
   nonFunctionalProperties
      dc#description hasValue "The output is a GetBNQuoteResult"
    endNonFunctionalProperties
    definedBy
      ?response memberOf bnPrice#GetBNQuoteResponse.

interface _"http://www.ntnu.no/goals/BarnesNobleGoal#interface"
   importsOntology
   _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"
  choreography barnesNobleChoreography
   stateSignature bnStatesignature
      importsOntology
         _"http://www.ntnu.no/ontologies/bnpriceOntology
         #barnesNoblePriceOntology"
```

```
    in
       concept bnPrice#GetBNQuote  withGrounding
          _"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#
          wsdl.interfaceMessageReference(bnprice/GetBNQuote/in0)"

    out
      concept bnPrice#GetBNQuoteResponse withGrounding
          _"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#
      wsdl.interfaceMessageReference(bnprice/GetBNQuoteResponse/out0)"

  transitionRules
    _"http://www.ntnu.no/ontologies/BarnesNobleGoal#transitionRules"
       forall {?request} with
       (?request memberOf bnPrice#GetBNQuote
      ) do
        add(_#1 memberOf bnPrice#GetBNQuoteResponse)
   endForall
```

B Java Code

In this section the source code for the two classes **BarnesNobleAdapter.java** and **BookPriceSearch.java** is included.

B.1 BarnesNobleAdapter.java

```
package main.masterthesis.adapter;

import ie.deri.wsmx.adapter.Adapter;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.io.StringReader;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Set;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.soap.SOAPElement;

import org.apache.log4j.Logger;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.omwg.ontology.Concept;
import org.omwg.ontology.Instance;
import org.wsmo.common.IRI;
import org.wsmo.execution.common.nonwsmo.WSMLDocument;
import org.wsmo.execution.common.nonwsmo.grounding.EndpointGrounding;
import org.xml.sax.InputSource;

import com.isoco.dip.adapter.XMLTranslator;

/**
 * Adapter used when communicating with the Abundant Technologies
 * Barnes & Noble book price search web service.
 * Contains methods for lowering and lifting to and from WSML and XML.
 *
 *
 * @author Øyvind Skytøen
 */

public class BarnesNobleAdapter extends Adapter {
   protected static Logger logger =
         Logger.getLogger(BarnesNobleAdapter.class);

   public BarnesNobleAdapter(String id) {
      super(id);
   }

   public BarnesNobleAdapter() {
      super();
```

```
}

public SOAPElement getHeader(Instance instance) {
   return null;
}

/**
 * This method lowers the information in the goal WSML to an
 * XML that can be sent to the Web service.
 *
 * @param instance
 *            the Instance created from the goal-input in the WSML.
 * @return the XML to be sent to the Web service
 */
public org.w3c.dom.Document getXML(Instance instance) {
   logger.debug("enter in getXML. Instance: " + instance);
   try {
      DocumentBuilderFactory factory = DocumentBuilderFactory
            .newInstance();
      factory.setNamespaceAware(true);
      DocumentBuilder builder = factory.newDocumentBuilder();

      Set concepts = instance.listConcepts();
      logger.debug("Concepts set:" + concepts);
      LinkedHashMap attributes = (LinkedHashMap) instance
            .listAttributeValues();
      Set names = attributes.keySet();
      Iterator iterator = names.iterator();

      if (concepts.size() != 1) {
         throw new RuntimeException(
               "The instance has more than one concept!");
      }
      Iterator iter = concepts.iterator();
      Concept concept = (Concept) iter.next();

      String iristring = concept.getIdentifier().toString();
      logger.debug("The instance belongs to: " + iristring);

      String sDoc = "";

      // a flag to check if the iri has been matched
      boolean validInstanceFound = false;

      if (iristring
            .equalsIgnoreCase("http://www.ntnu.no/ontologies/"
            + "bnpriceOntology#getBNQuote")) {

         validInstanceFound = true;
         sDoc += "<GetBNQuote xmlns=\"http://"
         +"www.abundanttech.com/webservices/BNPrice\">"
               + "\n";

         IRI att = (IRI) iterator.next();
         Set values = (Set) attributes.get((Object) att);
         String sISBN = values.iterator().next().toString();
         if (att.toString().equalsIgnoreCase(
               "http://www.ntnu.no/ontologies/bnpriceOntology#sISBN")) {
```

```
            sDoc += "<sISBN>" + sISBN + "</sISBN> \n";
         }
         sDoc += "</GetBNQuote>";
      }

      // the iri has not been matched
      if (validInstanceFound == false) {
         throw new RuntimeException("The type of the instance: "
               + iristring + " was not recognized. ");
      }

      InputSource is = new InputSource(new StringReader(sDoc));
      org.w3c.dom.Document doc = builder.parse(is);

      ByteArrayOutputStream stream = new ByteArrayOutputStream();
      OutputFormat format = new OutputFormat(doc);
      format.setIndenting(true);
      XMLSerializer serializer = new XMLSerializer(stream, format);
      serializer.serialize(doc);

      return doc;

   } catch (Exception e) {
      logger.error("Error in processing document");
      return null;
   }
}

/**
 * This method lifts the incoming XML to a WSML-representation of the
 * information in the XML. Uses a XMLTranslator with the XML and a
 * xslt file to do the transformation from XML to WSML.
 *
 * @param document
 *            The incoming XML
 * @param endpoint
 *            The EndpointGrounding (not used here)
 * @return a WSMLDocument created from the information in the XML
 */
public WSMLDocument getWSML(String document, EndpointGrounding endpoint) {
   WSMLDocument wsmlDocument = new WSMLDocument("");
   String initialXML = document;
   String translatedXML;
   String xslFileName = "barnesNoble.xsl";
   XMLTranslator translator = new XMLTranslator();

   InputStream xsltFile = null;
   try {
      // For some reason the reference to the file must include the folder
      // dist for the file to be found during unittests:
      // "dist/resources/resourcemanager/masterthesis/xslt/"
      xsltFile = new FileInputStream("resources" + File.separator
            + "resourcemanager" + File.separator + "masterthesis"
            + File.separator + "xslt" + File.separator + xslFileName);
   } catch (FileNotFoundException e) {
      e.printStackTrace();
   }
   // Replaces the result string "Price Not Available" to avoid parsing
```

```
      // error.
      initialXML = initialXML.replace("Price Not Available",
            "PriceNotFoundError");

      translatedXML = translator.doTransformation(initialXML, xsltFile);

      wsmlDocument.setContent(translatedXML);

      return wsmlDocument;
   }

}
```

B.2 BookPriceSearch.java

```
package ie.deri.webservices;

import ie.deri.wsmx.commons.Helper;
import ie.deri.wsmx.executionsemantic.ExecutionSemanticsFinalResponse;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;

import javax.management.MBeanServer;
import javax.management.ObjectName;

import org.omwg.ontology.Instance;
import org.omwg.ontology.Value;
import org.wsmo.common.Entity;
import org.wsmo.factory.Factory;
import org.wsmo.factory.WsmoFactory;

/**
 * This class defines the entry point for searching for book prices
 * based on a ISBN.
 *
 * @author Øyvind Skytøen
 *
 */
public class BookPriceSearch {

 /**
  * This method defines the operation getBookPrice. The Method creates a goal
  * WSML with an instance, where the value from the ISBN string in the input
  * is used. The goal is sent to the server, and the result is sent back to
  * the requester.
  *
  * @param sISBN
  *            the ISBN of a book.
  * @return a String with the price of the book that was searched for
  */
 public String getBookPrice(String sISBN) {
  String wsmlMessageGoal = "wsmlVariant "
    + "_\"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight\" "
    + "namespace { _\"http://www.ntnu.no/ontologies/BarnesNobleGoal#\","
    + "dc _\"http://purl.org/dc/elements/1.1/\", "
```

```
  + "xsd _\"http://www.w3.org/2001/XMLSchema#\", "
  + "bnPrice _\"http://www.ntnu.no/ontologies/bnpriceOntology#\"} "
  + "goal _\"http://www.ntnu.no/ontologies/BarnesNobleGoal\" "
  + "nonFunctionalProperties "
  + "dc#title hasValue \"Barnes & Noble price search goal with instance\""
  + "dc#format hasValue \"text/plain\" "
  + "dc#type hasValue _\"http://www.wsmo.org/2004/d2#goals\" "
  + "dc#language hasValue \"en-US\" "
  + "endNonFunctionalProperties "
  + "importsOntology {_\"http://www.ntnu.no/ontologies/bnpriceOntology#"
  + "barnesNoblePriceOntology\"} "
  + "capability _\"http://www.ntnu.no/goals/BarnesNobleGoal#capability\""
  + "importsOntology _\"http://www.ntnu.no/ontologies/bnpriceOntology#"
  + "barnesNoblePriceOntology\" "
  + "sharedVariables ?request "
  + "precondition bnPricePrecon "
  + "nonFunctionalProperties "
  + "dc#description hasValue \"The input has to be a getBNQuote\" "
  + "endNonFunctionalProperties "
  + "definedBy "
  + "?request memberOf bnPrice#GetBNQuote. "
  + "postcondition bnPricePostcon "
  + "nonFunctionalProperties "
  + "dc#description hasValue \"The output is a GetBNQuoteResult\" "
  + "endNonFunctionalProperties "
  + "definedBy "
  + "?response memberOf bnPrice#GetBNQuoteResponse. "
  + "interface _\"http://www.ntnu.no/goals/BarnesNobleGoal#interface\""
  + "importsOntology _\"http://www.ntnu.no/ontologies/bnpriceOntology"
  + "#barnesNoblePriceOntology\" "
  + "choreography barnesNobleChoreography "
  + "stateSignature bnStateSignature "
  + "in "
  + "concept bnPrice#GetBNQuote withGrounding "
   + "_\"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#"
   + "wsdl.interfaceMessageReference(bnprice/GetBNQuote/in0)\" "
  + "out "
  + "concept bnPrice#GetBNQuoteResponse withGrounding "`
  + " _\"http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#"
  + "wsdl.interfaceMessageReference(bnprice/GetBNQuoteResponse/out0)\""
  + "transitionRules _\"http://www.ntnu.no/ontologies/BarnesNobleGoal"
  + "#transitionRules\" "
  + "forall {?request} with "
  + "(?request memberOf bnPrice#GetBNQuote "
  + ") do "
  + "add(_#1 memberOf bnPrice#GetBNQuoteResponse) "
  + "endForall "
  + "ontology _\"http://www.ntnu.no/ontologies/getBNPriceOntology\" "
  + "importsOntology _\"http://www.ntnu.no/ontologies/bnpriceOntology"
  + "#barnesNoblePriceOntology\" "
  + "instance "
  + "newBNGoal memberOf bnPrice#GetBNQuote "
  + "bnPrice#sISBN hasValue \"" + sISBN + "\"";

System.out.println("Getting Servlet Context");

// get reference to MBeanServer
MBeanServer mBeanServer = (MBeanServer) Helper
```

```
    .getAttribute("MBeanServer");

  try {
   ObjectName commManagerName = new ObjectName(
     "components:name=CommunicationManager");
   //
   // check if registered
   boolean flag = mBeanServer.isRegistered(commManagerName);
   if (!flag) {
    return "error";
   }

   Object returnData = mBeanServer.invoke(commManagerName,
     "achieveGoalFullResponse",
     new Object[] { wsmlMessageGoal },
     new String[] { "java.lang.String" });
   ExecutionSemanticsFinalResponse esResponse =
    (ExecutionSemanticsFinalResponse) returnData;

   WsmoFactory wsmoFactory = Factory.createWsmoFactory(null);

   System.out.println("Recieved mesesages: "
     + esResponse.getReceivedMessages());
   if (esResponse.getReceivedMessages().size() != 1) {
    esResponse.getExecutionSemantic().cleanUp();
    return "PriceNotFoundError";
   }
   List<Entity> instancesResponse = Helper
     .getInstancesOfConcept(new ArrayList(esResponse
       .getReceivedMessages()),
       "http://www.ntnu.no/ontologies/bnpriceOntology#GetBNQuoteResponse");
   Set attrValues = ((Instance) instancesResponse.get(0))
     .listAttributeValues(wsmoFactory
       .createIRI("http://www.ntnu.no/ontologies/"
        +"bnpriceOntology#GetBNQuoteResult"));

   Value v = ((Value) attrValues.toArray()[0]);

   // clean up after communication
   esResponse.getExecutionSemantic().cleanUp();

   return v.toString();

  } catch (Exception e) {
   e.printStackTrace();
   return "Error";
  }
 }

}
```

C XSL for Lifting Adapter

Below is the XSL code used for the lifting from XML to WSML in the lifting adapter.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ws="http://www.abundanttech.com/webservices/BNPrice"
 xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.1">

 <xsl:output method="text" encoding="utf-8" indent="yes" />

 <xsl:template match="/">
  <![CDATA[
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://ww.ntnu.no/ontologies/" ,
bnPrice _"http://www.ntnu.no/ontologies/bnpriceOntology#",
wsmostudio _"http://www.wsmostudio.org#" }

ontology _"http://www.ntnu.no/ontologies/getBNQuoteResponseInstances"

importsOntology
 _"http://www.ntnu.no/ontologies/bnpriceOntology#barnesNoblePriceOntology"
   ]]>
  <xsl:apply-templates select="//ws:GetBNQuoteResponse" />
 </xsl:template>

 <xsl:template match="//ws:GetBNQuoteResponse">
   <![CDATA[
   instance getBNQuoteResponse memberOf bnPrice#GetBNQuoteResponse
   ]]>
   <![CDATA[
   bnPrice#GetBNQuoteResult hasValue
   ]]>
   <xsl:value-of select="ws:GetBNQuoteResult" />
 </xsl:template>

</xsl:stylesheet>
```

D XML for BookPriceSearch Entry Point

Below is the XML used in building of the BookPriceSearch entry point.

```
<service name="BookPriceSearch" scope="application">
   <description>
      Entry point for a book price search based on ISBN.
  </description>
  <messageReceivers>
   <messageReceiver
      mep="http://www.w3.org/2004/08/wsdl/in-only"
    class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
    <messageReceiver
      mep="http://www.w3.org/2004/08/wsdl/in-out"
    class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </messageReceivers>
  <parameter name="ServiceClass">
   ie.deri.webservices.BookPriceSearch
   </parameter>
</service>
```

E Attachments

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

E.1 Source Code

The Java code that was created for the SWS implementation is included in the *Source Code* folder. **BarnesNobleAdapter.java** contains the code for the adapters, and **BookPriceSearch.java** contains the code for the entry point. **Invoker.java** is also included because changes have been made to it that is not in the public SVN repository. **Service.xml** is the file created to describe the new entry point to WSMX, and **build.xml** is the Ant build file for the Web service component. The Ant build file has been altered to include the new entry point.

E.2 Binary Distribution

The files for the compiled WSMX server is located in the *WSMX Distribution* folder. The ontology descriptions for the SWS implementation, and the XSL file for the lifting adapter, can be found here in the folder */resources/resourcemanager/masterthesis/*. Use the file **startc.bat** to start the server. The WSDL for the entry point to the SWS can be found at http://localhost:8050/axis/services/BookPriceSearch?wsdl. Use a tool like for instance SoapUI [30] to connect to this WSDL, and send requests to the server with an ISBN. Table 4 contains a few examples of ISBNs for books where the price is available.

Table 4: Examples of Books with their ISBNs

Book	**ISBN**
Executing SOA: A Practical Guide for the Service-Oriented Architect	0132353741
Web Services: Principles and Technology	0321155556
Semantic Web Services: Concepts, Technologies, and Applications	3540708936
Semantic Web Services, Processes and Applications (Semantic Web and Beyond)	0387302395

[30] http:// www.soapui.org/