Mediation in the context of Semantic Web services: resolving heterogeneities in B2B scenarios

Thesis

How to cite:

For guidance on citations see FAQs.

© [not recorded]

https://creativecommons.org/licenses/by-nc-nd/4.0/

Version: Version of Record

Link(s) to article on publisher’s website:
http://dx.doi.org/doi:10.21954/ou.ro.0001005d

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online’s data policy on reuse of materials please consult the policies page.
Mediation in the Context of Semantic Web Services: Resolving Heterogeneities in B2B Scenarios

A dissertation submitted in partial completion of the requirements for the degree Doctor of Philosophy in Computer Science

Liliana Cabral

KMi

The Open University, UK

Submitted – 03-11-2008
Acknowledgements

It has been an amazing journey to complete this thesis while working at KMI in the Open University, UK. As a part-time PhD student this journey had further challenges, since it required coping with two working frames, and also finding spirit to literally advance a little bit every day. This thesis spanned through the course of two European Integrated Projects, namely DIP and SUPER in which I have participated. These projects inspired the ideas and offered the context for the development of the novel technologies presented in this research work. However, completing this work also required the support of many people to whom I am very grateful.

I would like to thank my supervisor, Prof. John Domingue for the technical and psychological support throughout the thesis, due greatly to his remarkable ability to listen. I am also grateful for having had Prof. Enrico Motta as my second supervisor, whose brilliant foundational research inspired the core of my thesis. I would like to thank Dr. Stefania Galizia and Dr. Alessio Gugliotta as colleagues for their early collaboration and discussions on my topic of research leading to common published papers. I would like to thank my close friends who brought life and happiness to my hard days in the last five years through fantastic parties, country hiking and trips around Europe, whenever I allowed a break. I would like to thank my parents, brothers and sisters now in my distant Brazil, just for keeping together this fundamental part of what I am today. I would like to thank my husband Georgios for always being there in the good and bad times. Finally, I thank my son Alan whose presence always brings balance to my life.
Related Publications

Several parts of this work have been published elsewhere. In particular, content from Chapter 2, Chapter 3, Chapter 4 and Chapter 6 has been published in:


There are also parts of this work that has been made public through deliverables in the DIP (Data, Information and Process Integration with Semantic Web Services) and SUPER (Semantics Utilized for Process management within and between Enterprises) IST projects as in the following.

- Deliverable D4.6. BPMO to sBPEL Translation. SUPER Project.


- Deliverable D5.3 – Requirements for Data and Process Mediation. DIP Project.
• Deliverable D5.4 – Business Data and Process Mediation Prototype. DIP Project.

• Deliverable D9.2 – Requirements for e-government Case Study. DIP Project.
Abstract

Semantic Web Services are Web Services enriched with ontological descriptions of their capabilities. We can use Semantic Web Services to build business applications on the Web or to compose them into business processes. However, ontological models for Semantic Web Services that take into consideration mediation aspects are scarce and underspecified. Consequently, so far there is no mediation framework which takes into account semantic descriptions of mediation aspects for coping with heterogeneous data and processes in the context of Semantic Web Services.

Our hypothesis is that a knowledge-based modelling approach to mediation can provide seamless integration of applications and semantic interoperability of data and processes. In this work we propose a knowledge-based model and approach to the mediation of Semantic Web Services, which extends the Internet Reasoning System (IRS) platform and applies the Web Service Modelling Ontology (WSMO). In addition, we propose the Business Process Modelling Ontology (BPMO) for supporting mediation of workflow-based processes based on Semantic Web Services.

Finally, we present three use cases in which different requirements for mediation are dealt with in order to validate our models and framework in business-to-business (B2B) application scenarios. We claim that our mediation approach is successful in bridging the gap between SWS requesters, who provide business-level descriptions of their tasks, and SWS providers, who supply execution-level descriptions of how they can solve tasks.
## Contents

Acknowledgements ........................................................................................................ ii

Related Publications .................................................................................................. iii

Abstract ..................................................................................................................... vi

Contents .................................................................................................................... vii

List of Figures .......................................................................................................... xiv

List of Tables ........................................................................................................... xvi

Chapter 1 Introduction ............................................................................................. 17

1.2 Research Questions ............................................................................................ 27

1.3 Research Methodology and Structure of the Thesis ........................................ 29

1.4 Contributions ..................................................................................................... 31

Chapter 2 Context and Related Work ....................................................................... 35

2.1 Introduction ........................................................................................................ 35

2.2 Web Services .................................................................................................... 36

2.3 Ontologies .......................................................................................................... 42

2.4 The Semantic Web ............................................................................................ 45
3.2.3 SWS Architecture Components .................................................................84

3.3 Implementing the SWS Framework ...............................................................85

3.3.1 The IRS-III approach ...............................................................................86

3.3.2 IRS-III Design Principles .......................................................................88

3.3.3 The IRS-III Service Ontology .................................................................90

3.3.4 The IRS-III Architecture .......................................................................93

3.4 Overview of the Languages OCML and WSML ........................................98

3.4.1 OCML – Operational Conceptual Modelling Language .....................99

3.4.2 WSML – Web Service Modelling Language .......................................102

3.4.3 OCML versus WSML ...........................................................................106

3.5 Discussion and Conclusions .......................................................................107

Chapter 4 A SWS Mediation Framework .........................................................109

4.1 The Problem of Mediation .......................................................................109

4.2 A Knowledge Modelling based Approach to Mediation ......................112

4.3 A Specification for Mediators within SWS ............................................114

4.3.1 Runtime Mediated Discovery ...............................................................120
4.3.2 Runtime Mediated Composition ................................................................. 122

4.4 Requirements for a SWS Mediation Framework ......................................... 122

4.5 Implementation of the Mediation Framework .............................................. 125

4.5.1 The Mediator Model .................................................................................. 126

4.5.2 Data and Process Mediation .................................................................... 135

4.5.3 Architecture .............................................................................................. 136

4.5.4 Conclusions about the IRS-III Implementation ....................................... 139

4.5.5 Mediator Model Reusability ................................................................. 140

4.6 A SWS Application Development Approach ............................................ 144

4.7 Discussion and General Conclusions ......................................................... 148

Chapter 5 Mediation and Business Process Modelling .................................... 155

5.1 Introduction .................................................................................................. 155

5.2 Approach .................................................................................................... 159

5.3 BPMO Description ...................................................................................... 163

5.4 A Translator for BPMO ............................................................................. 170

5.4.1 Mapping BPMO to Semantic BPEL ..................................................... 171
5.4.2 Translation Approach and Implementation ............................................178

5.5 Discussion and Conclusions ..........................................................................187

Chapter 6 Evaluation..................................................................................................192

6.1 Introduction......................................................................................................192

6.2 Use case in the e-government domain .............................................................193

6.2.1 Requirements for Service-Oriented Application Integration ....................195

6.2.2 Our Solution..............................................................................................196

6.2.3 Results.......................................................................................................206

6.3 Use Case in the Telecommunications domain .................................................209

6.3.1 Requirements for Business Process Modelling.........................................212

6.3.2 Our Solution..............................................................................................213

6.3.3 Results.......................................................................................................222

6.4 The SWS challenge..........................................................................................222

6.4.1 Requirements for Heterogeneous Message Exchanges .............................224

6.4.2 Our solution ..............................................................................................224

6.4.3 Results.......................................................................................................233
Appendix C IRS-III Internal components for SWS Discovery ........................................276

Appendix D IRS-III Internal components for SWS Orchestration............................278

Appendix E The Business Process Modelling Ontology...........................................280

Appendix F BPMO2SBPEL Translator Rules...........................................................299

Appendix G Semantic Descriptions used in the Use Case on E-government............327

Appendix H Semantic Descriptions used in the Use case on Telecommunications..352

Appendix I Semantic Descriptions used in the SWS Challenge Use case ...............359
List of Figures

Figure 2-1. Web Service Usage Scenario ..........................................................38

Figure 2-2. Web Services enabling standards....................................................39

Figure 2-3. Semantic Web Enabling standards...................................................45

Figure 2-4 IRS-II Architecture...........................................................................50

Figure 2-5 IRS-II Browser snapshot...................................................................53

Figure 3-1 The IRS-III Architecture...................................................................93

Figure 4-1 Illustration of cases for Mediation in the context of SWS....................116

Figure 4-2 An OO-mediator diagram ...............................................................128

Figure 4-3 A WG-mediator diagram...............................................................131

Figure 4-4 A sample GG-mediator .................................................................134

Figure 4-5 Mediation framework of IRS-III......................................................137

Figure 4-6 Mappings Ontology.........................................................................142

Figure 4-7 A simplified SWS brokering scenario using IRS-III .........................145

Figure 4-8 A generic SWS application architecture...........................................146

Figure 5-1 BPMO Model Main Elements.........................................................159
Figure 5-2 Partial set of the SUPER Business Process Ontologies...........................161

Figure 5-3 ATL Transformation of WSML instances ........................................179

Figure 5-4 Example of BPMO instance in XML WSML.......................................180

Figure 6-1 Screen Shot of IRS-III Browser with SWS descriptions ......................197

Figure 6-2 User Interface screen shot of the e-government application....................198

Figure 6-3 Illustration of IRS-III orchestration and mediators.............................202

Figure 6-4. Business view of the Digital Content Download process ......................211

Figure 6-5 BPMO Diagram of the Content Provision Process................................214

Figure 6-6 Query result: all goal tasks in the Content Provision process...............220

Figure 6-7 Query result: URI of a goal in the Content Provision process...............221

Figure 6-8 BPMO diagram of the Moon Mediator process.....................................226

Figure 6-9 Query Result using OrderFromPurchaseOrderRequest Mapping...........231

Figure 6-10 Query Result for Finding Workflow Activities ....................................232
List of Tables

Table 2-1. Delivered components of existing SWS approaches.................................61

Table 5-1 Description of BPMO elements and correspondence to BPMN ...............164

Table 5-2 Description of BPMO elements and the mappings to sBPEL elements....172
Chapter 1  Introduction

“The roads are for journeys, not destinations.” Buddha

This thesis is about Mediation in the context of Semantic Web Services, in particular when Semantic Web Services are applied to Business-to-Business (B2B) scenarios. We propose a view on how Mediation can be defined and implemented, essentially investigating the role of Mediation and its semantic representation in the context of Semantic Web Services. Therefore, this thesis is ultimately about the use of ontology-based models and their role in the semantic interoperability of business data and processes.

Semantic Web Services for Business-to-Business Applications

The Semantic Web Services vision exploits the idea of using ontologies for semantically describing services in order to enable their automated discovery, composition and execution. In particular, this vision applies to business applications that require interoperating across domains and organisations. Business organisations need flexibility to work with new business partners – service requesters and service providers - and agility when market requirements change, therefore, they can use Semantic Web Services to expose process functionality as services as well as to represent service capabilities and requests at the semantic level in order to interoperate with other businesses.
The Mediation Problem

However, as service providers and service requesters may be independent, mediation is required to solve mismatches between different data formats and process behaviours. For example, in a business scenario where an online book seller (e.g. Amazon.com) has to interact with different book suppliers, the involved partners must not only adhere to the communication standards (e.g. WSDL and SOAP) but also comply with database schema (e.g. customer_id, book_ref, etc.) as well as the operation invocation order (e.g. login, search, buy, etc.) of their provided services. Although the advent of Web Services made the integration of services a far easier task, that still requires a request to be mapped to the interface requirements of available services.

At the technical level, a software developer would typically follow several steps in order to implement mediation for business applications: a) checking whether two Web services or processes are compatible, which means that the messages exchanged should match; b) deciding how to handle incompatibilities, which typically involves building a mediator between the processes (otherwise changing the processes); c) deciding how to implement the mediation, which would involve a combination of techniques to generate data mappings and to combine business activities from the participant processes; and d) implementing the applications according to the previous steps using a compliant execution environment. In this thesis we will overlook step (a) since mediation is in fact applied after this step in case of existing incompatibilities, and focus on the remaining steps, assuming that mediation will be needed and
supported from the viewpoint of semantic interoperability in the context of Semantic Web Services.

From the scenario above we define mediation in the context of Semantic Web Services as the task of resolving data and process mismatches during the interaction between a service provider and a service requester using ontological service descriptions.

**Interoperability and Semantic Web Services**

The new generation World Wide Web using innovative technologies has enabled people and machines alike to exchange information and services in the networked knowledge society. This is possible due to a number of Web standards supporting the interoperability of data formats (e.g. XML\(^1\)), data communication (e.g. SOAP\(^2\)), Web Service interface description (e.g. WSDL\(^3\)), and data semantics (e.g. RDF\(^4\), OWL\(^5\)).

Semantic Web Services is one of the emergent technologies built on these standards, which provide the semantic interoperability of Web Services by combining Semantic Web (Berners-Lee, 2001) and Web Service (W3C, 2001) technologies.

Web Services are reusable software components which can be invoked over the Web via standardized data and communication protocols. A Web Service can be accessed

\(^1\) http://www.w3c.org/\!/XML/\!Schema/
\(^2\) http://www.w3c.org/TR/SOAP/
\(^3\) http://www.w3c.org/TR/WSDL/
\(^4\) http://www.w3c.org/RDF/
\(^5\) http://www.w3c.org/OWL/
through its exposed interface, described using standards such as WSDL. Web Services setup a new paradigm for the way software programs are developed since they can be composed to create new applications.

The Semantic Web is concerned with the definition of ontology languages, reasoning engines and the development of supporting tools for describing Web resources. The resulting infrastructure can support publishing and resolving ontological descriptions of terms and concepts. In addition, it provides the necessary techniques for reasoning about these concepts, as well as resolving and mapping between ontologies, thus enabling semantic interoperability through the identification and mapping of semantically similar concepts.

Therefore, mapping mechanisms between distributed ontologies are required to solve semantic heterogeneity. Various approaches from the ontology matching research area (Noy, 2000; Maedche et al., 2002; Kotis et al., 2004; Euzenat, 2004) deal with the problems of: a) lifting syntactic data standards to ontologies; b) finding similarities between ontology elements (e.g. concepts, properties and relations); c) creating mappings (semantic bridges) between ontology elements; and d) executing these mappings.

In this thesis our focus is on (c), and (d) above since we are more concerned about modelling mediation aspects that can support Semantic Web Service activities than the automation of matchmaking. That is, we are interested in using mappings for bridging between different service ontologies. We want to explore the use of ontologies for mediation because when meta-data is represented at the syntactic level
such as in XML Schemas, matching of data is possible at the lexical and structural level, but when meta-data is represented at the semantic level using ontologies, further techniques for matching and generating mappings can be applied. For example, inputs in service interfaces that do not match at the syntactic level can be mapped at the semantic level, thus facilitating discovery or composition of services. Also mappings can be applied when incompatible services must interact.

As a combination of Semantic Web technologies and Web Service technologies, Semantic Web Services inherit the capacity to achieve semantic interoperability with ontologies, and the reusability of Web Services, but also inherit the heterogeneity problems that arise when two or more ontologies or two or more services have to be combined during business-to-business or enterprise application integration. Therefore, tackling the mediation problem of Semantic Web Services will provide seamless integration and semantic interoperability of data and processes.

As more and more businesses are offered through the Web, which require flexibility and agility in finding and combining the services of partner organisations, mediation becomes a priority. In particular, a significant amount of effort is spent mapping between different data formats and process behaviour of applications in business-to-business (B2B) scenarios. We aim at comprehensively supporting mediation between service requesters and service providers over the Web so that business applications can successfully interoperate with minimum human intervention.
**Traditional Mediation Approaches**

Traditionally, the notion of data mediation has been associated with the automation of data integration in order to overcome heterogeneities in information systems in areas such as enterprise application integration (Wiederhold, 1992). A typical approach for mediation in this context usually involves the introduction of a common unified representation of data (a meta-model) to which heterogeneous information resources are mapped. The data mediator executes the provided mappings using the unified representation when a retrieval request is invoked over the mapped resources. Through the mappings, the mediator performs tasks such as aggregation and disambiguation. Extended implementations of this approach consider ontologies as the common unifying representation (e.g. Sheth, 1998; Omelayenko, 2002) as well as Web Services as information resources (e.g. Liu, 2004). Thus, in this case, data mediation involves applying mappings between the common meta-level representation and the heterogeneous data formats.

Approaches to process mediation, on the other hand, is typically considered in the context of Web Service composition (Wohed et al., 2003), which can be defined as the task of finding suitable Web Services for fulfilling business process workflow activities. We refer to Process Mediation in this context when the process that executes the workflow is in fact controlling the invocation of Web Services from partner processes so that they can interact. That is, the modelling of the mediator process is based on the combination of control-flow, data-flow and atomic activities of two or more processes being mediated. In this context, mediation may be based on
more classical formalisms such as Planning (McIlraith et al., 2001), Petri-nets (Aalst
and Hee, 2002) and Process Algebra (Salaun et al., 2006); or based on business
process modelling standards such as BPEL (Andrews et al., 2003).

However, automatic approaches only consider the combination of perfectly matching
inputs and outputs (or messages) and sequential order of activities. Our stance in this
thesis is that we must consider mediation in the presence of data and process
heterogeneities. In particular, when composing services together there can be
mismatches in the format and order in which information is exchanged between a
requester and a provider. Thus, the mediator must handle incompatible service
interactions. Within business application scenarios, we argue that mappings between
service ontologies must be explicitly declared and completely accurate. This is
because these mappings have an integration purpose and will relate two or more
partner business entities for the purpose of exchanging information. The mappings
must be valid and accurate in order to be applied during the SWS invocation.

**Our Proposed Approach**

We propose a knowledge modelling approach to mediation in the context of Semantic
Web Services, which builds on requirements both for integrating services from
different providers and for mapping between different knowledge domains. We focus
our approach on system to system interactions instead of human to system interfaces,
thus facilitating the work of software developers.

Within this approach, we are less interested in algorithms which can detect whether
two representations match (syntactically or semantically) and more interested in
finding ways to explicitly represent mediation aspects which facilitate the integration of data and processes. We exploit the ability to provide mapping rules between domain ontologies as well SWS descriptions, so that we bridge the gap between representations in different levels. For example, in Semantic Web Services we can bridge between the requester representation of a service (a Task or Goal) and the provider representation (a Web Service); likewise, in business process modelling, we can bridge between the business-level representation of a process (the business analyst view) and the executable representation of a process (the programmer view). In our view Semantic Web Services are Web Services that use a conceptual model by means of ontologies to describe domain knowledge as well as service capabilities. We can use Semantic Web Services to build business applications on the Web or compose them into business processes.

We adopt a conceptual model for mediation in order to describe in a machine understandable way the types of mediators as well as the types of mappings needed for solving mismatches at the semantic level. In this way, mediation within Semantic Web Services can be applied for composing, selecting and interacting with applicable Web Services. A Semantic Web Service broker can be used at runtime for solving mismatches between Web Service requesters and providers using the provided conceptual model. In particular, we investigate the use of mediators both from the infrastructure viewpoint and from the development viewpoint. The former allows for the design and implementation of architectural components for handling mediation. The latter is necessary for complying with the mediation requirements of applications built from Semantic Web Services.
We aim at providing an approach and implemented framework for mediation that follows on the work on reusable components for knowledge modelling (Motta, 1999; Omelayenko et al., 2003). Consequently, our contributions include the partial development of the IRS-III framework, extending IRS-II (Motta, 2003) and applying the WSMO conceptual model (Fensel et al., 2006). Our proposed mediation framework is implemented as part of IRS-III, integrated with other components, such as the library of reusable components.

In addition, as part of a more generic solution to Process Mediation, we develop the Business Process Modelling Ontology (BPMO), which is a high-level representation for business process workflows. BPMO uses Semantic Web Services to represent interaction tasks and mediation tasks. Moreover, we develop a translator which can transform these business-level process models into executable process models, thus bridging the gap between the business view and the developer view. We believe our approach will ultimately facilitate the decision making of business analysts, and make it easy for software developers to reuse and integrate mappings and services into new software applications.

Thus, the goal of this thesis is to provide a knowledge modelling based approach to mediation that will bring forward the state-of-art of Semantic Web Services, making it easy for developers to contribute with mappings as well as new services onto integrated software systems, in order to enable companies to achieve their time-to-market objectives.
Use cases

We will evaluate our approach empirically through three different use cases: a use case in the e-government domain (within the DIP project\(^6\)), which will allow us to demonstrate the expressiveness and adaptability of our models and also how to build applications from Semantic Web Services; a use case in the telecommunication domain (within the SUPER project\(^7\)), which will focus on process mediation and demonstrate the use an ontology for composing services into business process workflows; and a use case taken from the Semantic Web Service Challenge\(^8\), which in particular presents a mediation scenario that can be used to discuss approaches to solve Semantic Web Service usage activities such as mediation.

Regarding our first use case, e-government has been one of the main application areas in which changes in policies and goals are associated with the use of innovative Web technologies. A number of projects in Europe have been initiated for example on innovative and better use of information technology to improve services and internal business in councils (for example see [http://www.localegov.gov.uk/e-innovations/](http://www.localegov.gov.uk/e-innovations/)). Governmental agencies need to communicate common information about citizens and services to one another, but their information systems are implemented over heterogeneous computing platforms and data formats with no central control. In this context mediation is needed in a number of cases: systems must be able to match the goals of citizens to the functionalities of services provided; developers must be able to

\(^6\) [http://dip.semanticweb.org](http://dip.semanticweb.org)

\(^7\) [http://www.ip-super.org](http://www.ip-super.org)

\(^8\) [http://sws-challenge.org](http://sws-challenge.org)
compose services to accomplish a complex task; mappings must be provided among heterogeneous services.

Our second use case is in the telecommunication domain and applies Semantic Business Process Management (SBPM) (Hepp et al., 2005). We start by providing semantic descriptions of organisational aspects, services and processes, and use these descriptions to progressively obtain an executable process. In this context, we exploit the fact that a business process can be exposed as a Semantic Web Service and make use of Semantic Web Services for representing business activities. There are a number of requirements that a business process analyst needs to fulfil. First, business analysts need to query the process space at the business level using their own terms and to reuse existing processes. Second, often the services from business partners cannot be excluded because of data and process mismatches, which should be solved. Finally, business processes need to be translated from the business level, where they are represented in a diagrammatic form, to the execution level, where they are represented in a standardized execution language (Andrews et al., 2003). We demonstrate our solution through a use case application scenario, which addresses the modeling and querying of processes and then the translation to executable business process workflows.

1.2 Research Questions

Research in the area of Mediation of Semantic Web Services is as new and foundational as research in the area of SWS. In this thesis we will investigate both how to extend ongoing work on SWS and how to apply existing work on Mediation
so that we can define and support mediation of SWS. The main issues are related to the use of semantic descriptions for mediating activities such as discovery, composition and invocation of web services, taking into account the requirements of business applications for handling data and process heterogeneity. Therefore, the main research question investigated in this thesis is "How can data and process heterogeneities be solved in the context of Semantic Web Services?". We break down this question into the following questions:

A) How can mediation at the semantic level improve the interoperability of Web Services?

We investigate the use of a first class mediator conceptual model as part of the service ontologies used in Semantic Web Services. We investigate the use of a SWS broker for mediating between a service requester and a service provider. This requires characterizing a framework for Semantic Web Services in a comprehensive way. The proposal of a mediation framework is presented within the context of a chosen Semantic Web Service infrastructure.

B) What is the role of mediation within Semantic Web Services?

We view mediation as another activity within Semantic Web Services usage. At the implementation level, we break mediation down into a number of components with specific roles during the life-cycle of Semantic Web Services.
C) What activities in Semantic Web Services need mediation?

We consider the fact that mediation is not a standalone activity. More specifically, mediation of data and processes is necessary when composing Web Services or selecting Web Services which must match specific user goals. We define the problem of mediated discovery and mediated composition.

D) What types of mismatches can be handled by mediation components?

Having defined where mediation can occur within Semantic Web Services, we investigate how to use semantic descriptions for solving specific types of conceptual mismatches. In addition, we analyse the problem of classifying types of mappings that can occur between services and propose a technique to support matchmaking.

1.3 Research Methodology and Structure of the Thesis

Our methodology is motivated by a grounded approach to applying technologies to real world cases. To summarize, we have extended an existing framework for SWS, which employs Knowledge Modelling techniques, defined our mediation approach accordingly, and applied this approach in various use cases. Our thesis is structured as follows.

Chapter 1 – This is the current chapter. We delineate the context of our research, characterize the problem of mediation within this context and outline our solution. In addition, we present our research questions and an outline of our contributions.
Chapter 2 – We describe the context of our work and state of the art of relevant technologies. We give examples of the use of Web Services and Semantic Web technologies. We then evaluate ongoing research in the area of Semantic Web Services and provide a comparison of relevant approaches. In addition, we analyse existing work on mediation of information systems and identify a list of research problems that we will undertake.

Chapter 3 – We formulate a consolidated characterization for Semantic Web Services based on the comparison of existing technologies and approaches given in the previous chapter, defining a conceptual framework for Semantic Web Services according to a service ontology, usage activities and execution components. Then, we provide a description of the IRS-III approach, which follows on this conceptual framework.

Chapter 4 – We define the task of mediation in the context of Semantic Web Services, and then describe our specification and proposed approach to mediation as part of the IRS-III. More precisely, we implement a Mediation Framework, which defines components and mechanisms for handling mediation as part of different activities, such as discovery and composition. We also analyse the problem of supporting mediation under an application development perspective and as the next contribution propose an approach for developing applications from Semantic Web Services.

Chapter 5 – In this chapter we contribute with the specification of the Business Process Modelling Ontology (BPMO) and describe its use for supporting mediation of
workflow-based processes based on Semantic Web Services. In addition, we contribute with the implementation of a translator for transforming business-level process models using BPMO into executable process models.

Chapter 6 – We evaluate our approach empirically via three use cases: a) a use case in the e-government domain, in which we apply our mediation framework as implemented in IRS-III, and build a SWS-based application in order to validate the requirements for enterprise application integration through a specific business scenario; b) a use case in the telecommunication domain, in which we apply our approach to process modelling using BPMO and validate the requirements of SWS based workflow modelling and execution; and c) the SWS challenge, which presents a mediation scenario that we use to validate requirements of semantic interoperability and use of semantic data mappings.

Chapter 7 – We present our conclusions and discuss our contributions. In addition, we discuss various aspects of our solution and future work.

1.4 Contributions

The main contributions of this thesis are:

- A characterization for Semantic Web Services combined with mediation.

- A Mediation framework for Semantic Web Services as part of the IRS-III infrastructure, which includes.
Mediation for selection - mediators can connect and map between the goals and the services provided.

Mediation for composition – A workflow model for combining several Semantic Web Services.

Mediation between ontologies- mediators can provide mappings for aligning ontological elements (classes, relations and instances).

Handling of mapping rules – a mapping mechanism from the underlying reasoning language can be used for declaring mappings explicitly.

Handling of Mediation services – a mechanism for executing mediation services as Web Services for handling transformations.

- A specification of mediation for Semantic Web Services, which is based on reusable components for knowledge modelling, including a definition of mediated discovery and mediated composition accordingly.

- A technique for developing Semantic Web Services based applications, including a generic application architecture integrating mediation aspects.

- The specification of the Business Process Modelling Ontology – an ontology which integrates SWS with business process workflows, facilitating process mediation.
• A translator for the Business Process Modelling Ontology (BPMO) – a tool for transforming business-level process models into execution-level process models.
Chapter 2  Context and Related Work

In this chapter, we describe the context of our work and state of the art of relevant technologies. We give examples of the use of Web Services and Semantic Web technologies. We then evaluate ongoing research in the area of Semantic Web Services and provide a comparison of relevant approaches. In addition, we analyse existing work on mediation of information systems and identify a list of research problems that we will undertake.

2.1 Introduction

Service-oriented software applications relying on syntactic descriptions of data and service functionality have serious problems with development, flexibility and automation. First, developing applications based on Web Services requires the human user in the loop because of the lack of a machine understandable description of the capabilities of the services. Also, composing and integrating services is a complex task because new services have to be adapted in order to work with existing services. Second, by using syntactic interface descriptions such as WSDL, system developers are unable to represent client desires and service capabilities; those can only describe the structure of the messages exchanged. Finally, automated search for services is hampered because of ambiguities in the meaning of inputs and outputs, and hence the meaning of the messages exchanged between requesters and providers. Semantic Web Services are a recent research effort involving industry and academia to solve these problems. This effort builds on the specification of a semantic layer on top of current standards for Web service interoperation and the use of ontology languages and reasoners provided by the Semantic Web.
Web Services are reusable software components, which can be invoked over the Web through standardized data and communication protocols. A Web Service can be accessed through its exposed interface, described using standards such as WSDL. Web Services setup a new paradigm for the way software programs are developed since they can be used across organisations as well as composed to create new applications.

Semantic Web technologies provide formal languages to express and represent semantic descriptions; tools to annotate Web services with these descriptions; and querying and reasoning mechanisms for interpreting these descriptions. Thus, Web Services can be augmented with rich formal descriptions of their capabilities, such that they can be utilized by applications or other services without human assistance or highly constrained agreements on interfaces or protocols.

In the following we describe and give examples of the use of Web Services and Semantic Web technologies. We then evaluate ongoing research in the area of Semantic Web Services and provide a comparison of relevant approaches. In addition, we analyse existing work on mediation of information systems and provide a list of research problems that we will undertake. From this survey we trace the technological path leading to our approach for mediation within Semantic Web Services.

### 2.2 Web Services

Web Services technology has had an impact on the way applications are developed over the Web, since they can be used to integrate business operations, reduce the time...
and cost of development and maintenance, as well as promote reuse of code. By allowing functionality to be encapsulated and defined in a reusable standardized format, Web services have enabled businesses to share and exchange functionality with arbitrary numbers of partners, without having to pre-negotiate communication mechanisms or syntax representations. This is due to the fact that Web Services are based on standard communication protocols, which allow programs implemented in diverse platforms to syntactically interoperate. Discovery of services has been made possible through the use of registries for publishing information about Web Services, which has enabled developers to search for Web services by keyword or pre-defined categories.

A Web Service is a software program identified by an URI, which can be accessed via the internet through its exposed interface. The interface declares the operations available to the requester and the types of messages to be exchanged during the interaction with the service. For example, a Web service for calculating the exchange rate between two currencies would declare the operation getExchangeRate with two inputs of type string (for source and target currencies) and an output of type float (for the resulting rate). Although there are different ways to implement Web Services, we basically assume that they are implemented as Web applications and hence deployed in Web servers so that they can be invoked by any software program or other Web Services using Web protocols.

The common usage scenario for Web services (Figure 2-1) involves three software entities: the service requester, which invokes services; the service provider, which
responds to requests; and the service registry where services can be published or advertised. First, a software developer will implement and deploy a service on the Web as well as generate its interface description. Then, this service can be published in a registry using an appropriate tool or API. The registry holds information which can include: a profile of the service provider (e.g. company name and address); a profile of the service itself (e.g. name, category); and the URL of its interface description. The requester can query the registry about one or more desired services and if they are successfully found then the requester can combine them and bind to the providers through the information provided in the interface description for invocation.

Key to the interoperation of Web Services is the adoption of enabling standard protocols. As can be seen in Figure 2-2, several XML-based standards, which sit on top of the traditional Web protocol (HTTP, URI) have been specified to support the usage scenario above. SOAP\(^9\) is W3C’s recommended XML-data transport protocol.

---

\(^9\) http://www.w3c.org/TR/SOAP/
XML schema\textsuperscript{10} (XML-S) is the language used for defining the structure and types of XML data. WSDL\textsuperscript{11} is the W3C recommended language for describing a Web Service interface. WSDL describes Web services in terms of messages, operations, port types and bindings. Messages declare the types of the inputs and outputs of operations; they can be complex types defined using XML Schema. Port types declare the operations available with associated messages. The bindings declare the transport mechanism (usually SOAP) used by each port type. WSDL also specifies one or more network locations or endpoints at which the service can be invoked. UDDI\textsuperscript{12} is the main standard specifying a registry for Web Services, which integrates with WSDL and SOAP. The main objective of a registry is to make Web Services available, enabling their dynamic discovery and selection. BPEL4WS\textsuperscript{13} is a standard built on top of WSDL to enable the composition of Web Services based on workflows.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{web_services_enabling_standards.png}
\caption{Web Services enabling standards}
\end{figure}

\textsuperscript{10} http://www.w3.org/XML/Schema/
\textsuperscript{11} http://www.w3.org/TR/WSDL/
\textsuperscript{12} http://www.uddi.org
\textsuperscript{13} http://www.ibm.com/developerworks/library/ws-bpel
To illustrate some of these Web Service standards, we show how they have been used in a Web Service provided by Amazon.com\textsuperscript{14} for searching for books available from the website. In particular, we exemplify a specific operation for searching for a book by author (AuthorSearchRequest) (Listing 2-1).

Listing 2-1 shows a source code excerpt of a typical SOAP message, which contains an envelope with a heading and body. The heading describes the XML based conventions for namespace; and the body represents the structure of the message instance, which contains values for the operation inputs (author = king; page=1; mode=books etc).

Listing 2-1. Source code excerpt of the SOAP message for AuthorSearchRequest operation

```xml
<?xml version="1.0" encoding="UTF-8"?>
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <SOAP-ENV:Body>
    <namespl:AuthorSearchRequest xmlns:namespl="urn:PI/DevCentral/SoapService">
      <AuthorSearchRequest xsi:type="m:AuthorRequest">
        <author>king</author>
        <page>1</page>
        <mode>books</mode>
        <tag>webservices-20</tag>
        <type>lite</type>
        <dev-tag>your-dev-tag</dev-tag>
        <format>xml</format>
        <version>1.0</version>
      </AuthorSearchRequest>
    </namespl:AuthorSearchRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 2-2 shows a source code excerpt of the actual WSDL file, which contains the XML schema declaration of the AuthorSearchRequest operation described previously.

\textsuperscript{14} http://www.amazon.com/webservices
Listing 2-2. WSDL source excerpt for Author Search Amazon Web Service

```xml
<wsdl:definitions xmlns:typens="http://soap.amazon.com"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soap="http://schemas.xmlsoap.org/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
                  xmlns="http://schemas.xmlsoap.org/wsdl/"
                  targetNamespace="http://soap.amazon.com"
                  name="AmazonSearch">

<wsdl:types>
 <xsd:schema xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://soap.amazon.com">
 <xsd:complexType name="AuthorRequest">
 <xsd:all>
 <xsd:element name="author" type="xsd:string"/>
 <xsd:element name="page" type="xsd:string"/>
 <xsd:element name="mode" type="xsd:string"/>
 <xsd:element name="tag" type="xsd:string"/>
 <xsd:element name="devtag" type="xsd:string"/>
 <xsd:element name="sort" type="xsd:string" minOccurs="0"/>
 <xsd:element name="locale" type="xsd:string" minOccurs="0"/>
 <xsd:element name="keywords" type="xsd:string" minOccurs="0"/>
 <xsd:element name="price" type="xsd:string" minOccurs="0"/>
 </xsd:all>
 </xsd:complexType>
 </xsd:schema>
</wsdl:types>

<message name="AuthorSearchRequest" type="typens:AuthorRequest"/>
</message>

<message name="AuthorSearchResponse">
 <part name="return" type="typens:ProductInfo"/>
</message>

<portType name="AmazonSearchPort">
 <operation name="AuthorSearchRequest">
 <input message="typens:AuthorSearchRequest"/>
 <output message="typens:AuthorSearchResponse"/>
 </operation>
</portType>

<binding name="AmazonSearchBinding" type="typens:AmazonSearchPort">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="AuthorSearchRequest">
 <input>
  <soap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://soap.amazon.com"/>
 </input>
 <output>
  <soap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://soap.amazon.com"/>
 </output>
 </operation>
</binding>

<service name="AmazonSearchService">
 <port name="AmazonSearchPort" binding="typens:AmazonSearchBinding">
  <soap:address location="http://soap-eu.amazon.com/onca/soap3"/>
 </port>
</service>
</wsdl:definitions>
```

The XML schema in this WSDL file declares the complex type `AuthorRequest` with attributes (author of type string and so on), which is the type of the input.
AuthorSearchRequest of message with the same name. The Port type is called
AmazonSearchPort, which is bound to the operation AuthorSearchRequest.

The examples above illustrate the issues mentioned in the introduction of this chapter. There is very little that a program (or software agent) can infer about what a service does, how it does it and the applicable constraints by looking at the syntactic (WSDL) description. Thus, automation is hampered, requiring the software developer to be in the loop for deciding which services to select and compose for building an application. Also, the developer has to take care of incompatible data types as well as resolve how to map results into required values, for example, when the output of one service needs to be used as input of one or more services.

2.3 Ontologies

Ontologies can be seen as information technology artefacts for sharing and reusing knowledge. Ontologies have played a central role in building the Semantic Web as well as in integrating heterogeneous sources of information. In this thesis we adopt the following definition of ontology from (Gruber, 1993): “An ontology is a formal, explicit specification of a shared conceptualization”. In more details, an ontology is a conceptual model about a domain of discourse reflecting a commonly accepted understanding. This model contains unambiguous vocabulary definitions with computational semantics, which are machine understandable.

Traditionally, ontologies have been developed within the Knowledge Modelling research community in order to facilitate knowledge sharing and reuse (Gruber,
1991). Typical work on ontology formalisation is represented in (Guarino et al., 1998). Research on ontologies and their applications have produced a variety of modelling languages with differing expressivity and inference power. One relevant outcome from this community was the language DAML+OIL\(^\text{15}\) from which the first Web ontology language (OWL) was derived (see example below).

Currently, ontologies have been used for the semantic annotation, reasoning and matching of Web contents, which are now being applied to the implementation of the Semantic Web vision (as explained in the next section). Ontologies provide greater expressiveness when modelling domain knowledge than XML schemas and can be used to communicate knowledge between people and heterogeneous and distributed application systems.

For illustration purposes, Listing 2-3 shows an excerpt of the Wine ontology\(^\text{16}\) represented in OWL. This sample shows the basic elements of an ontology defined in OWL: classes, properties (relations) and the inheritance (subclassing) mechanism. This ontology defines the concept Wine using the OWL class construct, which uses RDF-S types. Also, this ontology defines the property MadeFromGrape and applies it to the class Wine, using the Restriction construct.

**Listing 2-3.** Excerpt of the Wine ontology in OWL

```xml
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Wine">
```

\(^{15}\) [http://www.daml.org/2001/03/daml+oil-index.html](http://www.daml.org/2001/03/daml+oil-index.html)

\(^{16}\) [http://www.w3.org/TR/owl-guide](http://www.w3.org/TR/owl-guide)
Lately, there has been extensive work relating ontologies with mediation; more specifically, work on ontology mapping approaches and matching algorithms. For example, the work of (Noy, 2000) is an early contribution to algorithms for the automation of ontology merging and alignment. (Kotis et al., 2004) presents a Description Logics (Baader et al., 2003) formalism for classifying degrees of matching between concepts in ontologies using inclusion and equivalence relations. (Euzenat, 2004) discusses reuse of ontology mappings and proposes an API for handling types of mappings between ontologies, called alignments, which are viewed as reusable objects that can be created and used by different users.

We can view the work above as representing the basic notion of data mediation as ontology mapping. That is, mappings are used to bridge between different ontologies. When meta-data is represented at the syntactic level such as in XML Schemas, matching of data is possible at the lexical and structural level, but when meta-data is represented at the semantic level using ontologies, further techniques for matching and generating mappings can be applied. Within Semantic Web Services, ontologies are primarily used for modeling semantic descriptions of Web services, which can be used to support mediation. For example, inputs in service interfaces that do not match at the syntactic level can be made compatible (same concept) or loosely compatible.
(e.g. subclass) at the semantic level, thus facilitating discovery. Also relation mappings can be applied when incompatibles services must interact.

### 2.4 The Semantic Web

The Semantic Web (Berners-Lee, 2001) is the vision of a Web of meaningful contents and services, which can be interpreted by computer programs. The goal is to provide semantic interoperability of information on the Web. From our perspective, the Semantic Web can also be seen as a vast source of knowledge, which can be modelled, shared and reused, so that users and machines alike will be able to match more accurately the information and services they need using the tools provided.

![Semantic Web Enabling standards](image)

Figure 2-3. Semantic Web Enabling standards

The Semantic Web is concerned with the definition of ontology languages and reasoning engines for describing Web resources as well as the development of supporting tools. This infrastructure can support publishing and resolving ontological descriptions of terms and concepts. In addition, it provides the necessary techniques for reasoning about these concepts, as well as resolving and mapping between
ontologies, thus enabling semantic interoperability through the identification and mapping of semantically similar concepts.

As with Web Services, Semantic Web enabling standards fit into a set of layered specifications (Figure 2-3) built on the foundation of URIs, XML, and XML namespaces. The current components of the Semantic Web framework are the RDF Core Model\textsuperscript{17}, the RDF Schema Language\textsuperscript{18} (RDF-S) and the Web Ontology Language (OWL)\textsuperscript{19}. These standards (submitted to W3C), build up a rich set of constructs for describing the semantics of online information resources. RDF is a standard for describing XML data through the representation of objects and their relations through properties. RDF-Schema is a simple type system, which provides information (metadata) for the interpretation of the statements given in RDF. OWL has been the first ontology language based on the Semantic Web, which promises to facilitate greater machine interpretability of Web content than RDF and RDF Schema by providing a much richer set of constructs for specifying classes and relations on top of them (see an example in the previous section).

Although OWL is currently the main XML-based ontology language for the semantic Web, other languages have been proposed and aligned with OWL for dealing with

\textsuperscript{17} http://www.w3c.org/TR/rdf-concepts
\textsuperscript{18} http://www.w3c.org/TR/rdf-schema
\textsuperscript{19} http://www.w3c.org/TR/OWL
specific types of applications. In particular, WSML\textsuperscript{20} (Web Service Modeling Language) has been specified for describing Semantic Web Services.

### 2.5 Semantic Web Service Approaches

In this section we introduce Semantic Web Services as the combination of Web Services technology and Semantic Web technology with the goal of providing the semantic interoperability of Web Services. We evaluate related and ongoing research in the area of Semantic Web Services and provide a comparison of relevant approaches.

Semantic descriptions of Web services are necessary in order to enable their automatic discovery, composition and execution across heterogeneous providers and domains. As shown in Section 2.2 existing technologies for Web services only provide descriptions at the syntactic level, making it difficult for requesters and providers to interpret or represent non-trivial statements such as the meaning of inputs and outputs or applicable constraints. This limitation may be overcome by providing a rich set of semantic annotations that augment the service description, hence defining Semantic Web Services.

To our knowledge, the seminal paper defining Semantic Web Services as the combination of Web Services and semantics is (McIlraith and Zeng, 2001). In this

\[\text{http://www.wsml.org}\]
paper, a Semantic Web Service is defined through a service ontology, which enables machine interpretability of its capabilities. The paper also contains a practical account of how to use the ontology for activities such as discovery and composition of services based on a knowledge based language (Golog) and planners, followed by a discussion on other usage such as monitoring.

We will analyse next the three main Semantic Web Service (SWS) approaches that are relevant to our work: IRS-II (Motta et al., 2003), which is the approach we extend in this thesis; OWL-S (Martin et al., 2004), which was the first Web-based approach and have influenced the specification of other SWS approaches; and WSMO (WSMO Working Group, 2005, Fensel et al., 2007), which provides a new conceptual model for SWS that we adopt in our approach. IRS-II (Internet Reasoning Service) is a knowledge-based approach to SWS, which evolved from research on reusable knowledge components (Motta, 1999). OWL-S (formerly DAML-S) is an agent-oriented approach to SWS, derived from OWL, providing fundamentally an ontology for describing Web service capabilities. WSMO (Web Service modeling Ontology) is a business-oriented approach to SWS, focusing on a set of e-commerce requirements for Web Services including properties such as quality of service and trust. These approaches are complementary in many ways and reflect different views on SWS. The following sections describe these approaches in more detail.

### 2.5.1 The IRS-II approach

The IRS-II approach to Semantic Web Services was derived from the IRS (Internet Reasoning Service) project (http://kmi.open.ac.uk/projects/irs), which had the overall
aim of supporting the automated or semi-automated construction of semantically enhanced systems over the Internet. IRS-I (Crubezy et al., 2003) supported the creation of knowledge intensive systems structured according to the UPML framework (Omelayenko et al., 2003). As in UPML, IRS-I distinguishes between the following categories of components (knowledge models) specified by means of an appropriate ontology:

- **Domain models.** These describe the domain of an application (e.g. vehicles, a medical disease).

- **Task models.** These provide a generic description of the task to be solved, specifying the input and output types, the goal to be achieved and applicable preconditions.

- **Problem Solving Methods (PSMs).** These provide abstract, implementation-independent descriptions of reasoning processes which can be applied to solve tasks in a specific domain.

- **Bridges.** These specify mappings between the different model components within an application.

IRS-II (Motta et al., 2003) extended IRS-I into a Semantic Web Service framework by integrating the above framework with Web Service technology, in particular connecting PSMs with deployed Web Services. More specifically, Tasks are used to represent the request for a Web Service, PSMs are used to represent the functionality of a Web Service, and domain models are used to represent the data associated with
these models. These are represented internally in OCML (Motta 1999), an Ontolingua-derived language (Gruber, 1993), which provides both the expressive power to express task specifications and service competencies, as well as the operational support to reason about these.

The main components of the IRS-II architecture (Figure 2-4) are the IRS-II Server, the IRS-II Publisher and the IRS-II Client, which communicate through the SOAP protocol. The IRS-II server holds semantic descriptions of Web Services (using the Tasks, PSMs and domain knowledge models). The IRS-II Publisher plays two roles in the IRS-II architecture. Firstly, it links deployed Web services to their PSM within the IRS-II server. Note that each PSM is associated with exactly one Web service although a Web service may map onto more than one PSM since a single piece of code may serve more than one function. Secondly, the publisher automatically generates a wrapper for code or WSDL files, which is used to invoke the deployed Web service. Once code is published within the IRS-II it appears as a standard message-based Web service, that is, a Web service endpoint is automatically

Figure 2-4 IRS-II Architecture
generated. There can be more than one type of Publisher or publishing platform, depending on the implementation of the service. This design option allows for instant deployment of code during publishing as explained before and brokering between the server and the actual code during invocation.

A key feature of IRS-II is that Web service invocation is capability driven. The IRS-II supports this by providing a task centric invocation mechanism. An IRS-II client program can simply request a task to be achieved and IRS-II locates an appropriate PSM and then invokes the corresponding Web service.

Listing 2-4 shows a Task description of the same book search operation example described previously (see WSDL source in Listing 2-2) for the Amazon.com Web Service. In task author-search-task, the input-roles values (e.g. tag, devt, etc) semantically represent the values defined in the input message of the AuthorSearchRequest operation in WSDL. Moreover, def-irs-soap-binding function is used to ground the semantic types to the corresponding SOAP types (also in the WSDL declaration).

**Listing 2-4 IRS-II Task description for Amazon’s Author Search Service**

```
(def-class author-search-task (goal-specification-task) ?task
  ((has-input-role
    :value tag
    :value devt
    :value authorsearch
    :value mode
    :value page
    :value format)
   (has-output-role :value result)
   (tag :type string)
   (devt :type string)
   (authorsearch :type string)
   (mode :type string)
   (type :type string)
   (page :type string)
   (format :type string)
   (result :type string)))
```
Listing 2-5 shows a corresponding PSM description for the Task in Listing 2-4, which in this case declares the same input roles. It also declares through the relation tackles-task-type which Task the WebService described by this PSM can solve.

Listing 2-5 IRS-II PSM description for Amazon’s Author Search Service

IRS-II was designed for ease of use. Developers can interact with IRS-II through the IRS-II browser (see Figure 2-5) as well as through a Java API, which allows a developer to view the knowledge models registered in IRS-II, edit service descriptions, publish and invoke individual services. Application programs can also use the Java API to combine tasks that can be achieved within an application scenario.

The snapshot of the IRS-II browser in Figure 2-5 shows a list of Tasks defined in a particular ontology (miakt-matching-task) with two pop-up windows; one for
receiving values for achieving a specific Task (find-images-matching-patient-task); and the other with the result of the invocation.

The core mechanism for mediation in IRS-II comes from the Bridge component of IRS (UPML), which covers semantic mappings amongst knowledge components (Crubezy et al., 2003). The process of building applications out of semantically described components involves several mapping activities, which a user has to undertake through an application configuration tool: mapping generic Tasks and PSMs to domain models, mapping PSMs to Tasks or, in general, adapting existing resources. More specifically, the knowledge components of a library can be
connected together in different running systems, through the creation of explicit mediating elements called bridge adapters by way of mapping relations between the ontologies of both components, such as: Task-Domain bridge; PSM-Domain bridge; and Task-PSM bridge. If the application domain knowledge does not conform to the Task ontology, the IRS tool supports users in constructing a mapping relation between the Task input and corresponding domain knowledge. A domain-Task mapping relation defines the transformation of a piece of domain knowledge or attributes into an instantiated input role for the Task; mappings may also be required for Task outputs to conform to the application domain ontology. The IRS supports the acquisition of domain-PSM mapping knowledge in a way similar to the domain-Task mapping during PSM configuration. A Task-PSM bridge maps the inputs and outputs of a selected PSM to the corresponding Task. IRS users specify the domain entities that fill-in the input–output roles of the PSM. Some of the roles for the PSM can be inherited from the configured Task. In addition, the selected PSM may define supplemental roles.

During invocation, the IRS first acquires application data from the user and instantiates the inputs of the Task and PSM by interpreting the domain–Task and domain–PSM mapping relations. The IRS also checks the preconditions of the PSM and Task on the mapped application data. The IRS then runs the code with the mapped inputs using Task–PSM mappings. Finally, the IRS fills-in the domain outputs with the results of PSM execution, possibly transformed with domain–PSM mapping relations.
In this thesis we will work on extending IRS-II into IRS-III and implementing mediation according to the requirements of business applications as presented in the next chapter.

2.5.2 The OWL-S approach

OWL-S\(^{21}\) (previously DAML-S) consists of a set of ontologies designed to describe and reason over service descriptions (OWL-S coalition, 2003). Unlike Web Service techniques, OWL-S originated from an AI background in the area of software agents, with the intention of supporting automated service discovery, composition, binding and execution, and thus minimizing human intervention, a necessary component when utilizing and orchestrating web services. The design of OWL-S was based on the concept of describing services in terms of data inputs/outputs, and knowledge preconditions and effects, thus facilitating the use of service descriptions as planning operators. This approach has previously been used to describe agent functionality within several agent systems and has been used with a variety of planners to solve higher level goals (McIlraith et al., 2001). AI planning uses a set of techniques to generate a process automatically based on the specification of a problem. These techniques attempt to use exploration methods such as searching, backtracking, and/or branching techniques in order to extract a solution.

OWL-S combines the expressivity of Description Logics (Baader et al., 2003), in this case from OWL, as well as pragmatism found in the emerging Web Services

\(^{21}\) http://www.w3.org/2002/ws/OWL-S
standards, to describe services that can be expressed semantically, and yet grounded within a well defined data typing formalism. It consists of three main upper ontologies: the Profile, the Process Model and Grounding. The Profile is used to describe services for the purposes of discovery; service descriptions (and queries) are constructed from a description of functional properties (i.e. inputs, outputs, preconditions, and effects - IOPEs), and non-functional properties (human oriented properties such as serviceName, etc, and parameters for defining additional meta data about the service itself, such as concept type or quality of service). In addition, the profile class can be subclassed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

OWL-S process models describe a process (or service) in terms of their constituent processes (or services). This is used both for reasoning about possible compositions (such as validating a possible composition, determining if a model is executable given a specific context, etc) and controlling the enactment/invocation of a service. Three process classes have been defined; the composite, simple and atomic process. The atomic process is a single, black-box process description with exposed IOPEs. Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world that must be asserted in order for an agent to execute a service. Effects characterize facts that become asserted given a successful execution of the service, such as the physical side-effects that the execution the service has on the physical world. The atomic process is the smallest element within the process model – this describes a single invokable element and is analogous to a WSDL operation. Typically, atomic processes refer to instances of processes
described by available services. Simple processes provide a means of describing process abstractions – such elements have no specific binding to a physical service, and thus have to be realized by an atomic process (e.g. through service discovery and dynamic binding at run-time), or expanded into a composite process. Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. These process workflows are constructed using a number of different composition constructs, including: Sequence, Unordered, Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, and Split+join.

The profile and process models provide semantic frameworks whereby services can be discovered and invoked, based upon conceptual descriptions defined within the Semantic Web (i.e. OWL) ontologies. The grounding provides a pragmatic binding between this concept space and the physical data/machine/port space, thus facilitating service execution. The process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types. Additional properties pertaining to the binding of the service are also provided (i.e. the IP address of the machine hosting the service, and the ports used to expose the service).

OWL-S was the first Web compliant ontology (i.e. using W3C standards) created for describing Web Services. It influenced the definition of other consecutively submitted standards and approaches. The original OWL-S working group
(http://www.daml.org/services/owl-s/1.0/) has been discontinued, but the approach as well as examples and tools are available from the W3C website.

It is worth noting, that OWL-S does not have any explicit construct or representation for supporting mediation or mappings of processes or services, which might imply that data mappings must be provided within domain ontologies. Also, required transformations of values for composite services must be performed by another process (or service).

2.5.3 The WSMO Approach

The Web Service Modeling Ontology\(^{22}\) – WSMO – is an approach to Semantic Web Services (WSMO Working Group, 2004), which provides a model for describing the various aspects related to Web services. This approach has been derived from WSMF (Web Service Modeling Framework) (Fensel and Bussler, 2002), which is centred on two complementary principles: the strong de-coupling of the various components that realize an e-commerce application; and the strong use of mediation in order to communicate in a scalable manner. Mediation is applied at several levels: mediation of data structures; mediation of business logics; mediation of message exchange protocols; and mediation of dynamic service invocation. WSMO is a formal service ontology which refines and develops WSMF, but also borrows from the work on Knowledge Modelling (Fensel et al., 1999; Fensel and Motta, 2001), in particular the UPML framework (Omelayenko et al., 2003).

---

\(^{22}\) http://www.wsmo.org
The WSMO approach comprises the work of three working groups: a) the WSMO working group (http://www.wsmo.org), which is responsible for specifying the meta-level concepts of the service ontology and applicable formalisms; b) the WSML working group (http://www.wsml.org), which is responsible for defining a family of languages and reasoners (Bruijn et al., 2006) for WSMO; and c) the WSMX working group, which is responsible for developing an execution environment for WSMO-based applications. WSMO and WSMX have recently been submitted to W3C and OASIS respectively. Some tools for editing WSMO have also been produced, such as WSMT (Web Service Modeling Tool) and WSMO Studio (Dimitrov et al., 2005).

The WSMO conceptual model consists of four main elements: ontologies, which provide the terminology used by other elements; goal descriptions, which define the problems that should be solved by Web services; Web services descriptions, which define various aspects of a Web service; and mediators, which are used to bypass heterogeneity problems. These elements have been implemented as part of the WSML language. Every Goal, Web Service, Mediator and Ontology definition is stored as a separate WSML file.

A Goal consists of the required capability, including precondition, assumption, post-condition and effect (all logical expressions), the required choreography and the required orchestration. Choreography describes how to communicate with a single Web Service in terms of message exchanges; and orchestration describes how to fulfil a functionality in terms of the composition of other Web Services. Web Service descriptions have a counterpart capability, choreography and orchestration
description. Choreography and Orchestration have been formalized as Finite State Machines in WSMO, however, in WSML choreography is supported by the definition of transition states, and orchestration supported by the definition of transition rules.

There are four types of mediators defined in WSMO:

- **OO-mediators** link ontologies to ontologies;

- **WW-mediators** link web services to web services;

- **WG-mediators** link web services to goals;

- **GG-mediators** link goals to goals.

An *OO-mediator* can be used by WSMO components when importing ontologies, especially for aligning, merging and transforming imported ontologies. A *GG-mediator* is used to specify goals by refining existing goals. It uses a logical expression to state the mappings between corresponding concepts in the information space (domain) of the goals. A *WG-mediator* is used to handle partial matches in goal-capability matching in web service discovery. In this case a logical expression is used to restrict the information space of objects passed between a goal and a web service to an exact match. A *WW-mediator* is used to connect two Web Services. They are intended to be used for choreography and orchestration of Web Services.

In this thesis, our proposed framework for mediation will be implemented as part of IRS-III, and will adopt the WSMO conceptual model.
2.5.4 Comparison and Conclusions about SWS Approaches

The comparison and conclusions in this section have been published in (Cabral, 2004) and are slightly updated here. Observe that work on WSMO (especially on WSMX) has evolved in the course of this writing. Also, the work on SAWSDL, although included in the previous sections for completeness, was not available by the time of this comparison. We compare the delivered results of IRS-II, OWL-S and WSMO and analyse the state-of-the-art of the SWS area. Another detailed comparison between OWL-S and WSMO can be found in (Lara, 2004).

Table 2-1. Delivered components of existing SWS approaches

<table>
<thead>
<tr>
<th>SWS Activities</th>
<th>IRS-II</th>
<th>OWL-S</th>
<th>WSMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Publishing</td>
<td></td>
<td>- Discovery</td>
<td>- Choreography</td>
</tr>
<tr>
<td>- Selection</td>
<td></td>
<td>- Composition</td>
<td>- Orchestration</td>
</tr>
<tr>
<td>- Task</td>
<td></td>
<td></td>
<td>- Mediation</td>
</tr>
<tr>
<td>Achievement</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td></td>
<td>- DAML-S Virtual Machine</td>
<td>- WSMX</td>
</tr>
<tr>
<td>Architecture</td>
<td>Server</td>
<td>- OWL-S</td>
<td>- WSMO</td>
</tr>
<tr>
<td>- Publisher</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Client</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Task/PSM Ontology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ontology</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1 shows the high-level elements of each approach as implemented by the time of this writing, which are given according to SWS activities, architecture and service ontology (extended in the next chapter).

The IRS-II approach has concentrated efforts in delivering a comprehensive infrastructure that clients can easily use from the description, to publishing, to the invocation of a Web Service or code. The main component of the IRS-II architecture is the server, which contains the ontology language and reasoner that implement operations over the knowledge-based models. Service invocations are atomic in IRS-
II and carried out via Task achievement, although a PSM can embody code for achieving subtasks. Also, selection is performed for finding which PSMs can solve a requested Task. The service ontology of IRS-II consists of the Task and PSM knowledge models, which separate the description of what a service does from the description of a particular implementation. Additionally, these models can also include a domain ontology for defining the types of inputs and outputs.

The main contribution of the OWL-S approach is its service ontology, which builds on the Semantic Web stack of standards. The OWL-S service ontology models IOPEs (inputs, outputs, pre-conditions and postconditions) to represent a Web service capability and includes a grounding which maps atomic services to WSDL descriptions for execution. Compared to the other approaches, OWL-S provides better defined constructs for representing composite services (process model). Since the OWL-S service ontology does not prescribe a framework implementation it has been used as the starting point of individual efforts towards SWS activities. There is some composition work developed based on its process model (Sirin et al., 2005, Wu et al., 2003). Discovery and matchmaking of Services have also been demonstrated, in particular, integrating with an UDDI registry (Paolucci et al., 2002). An early execution environment for DAML-S (preceding OWL-S) is the DAML-S Virtual Machine (Paolucci et al., 2003), which integrates a DAML-S processor (for the process model), a DAML inference engine, and an invocation component with an agent reasoning system.
The WSMO group focused on bringing business requirements into account when proposing a conceptual framework. In practice, WSMO combined the work of three working groups, consisting of the WSMO conceptual model (WSMO Working Group, 2004), the family of languages – WSML (Brujin et al., 2006) and the execution environment - WSMX. Some of the outcomes are still in the form of formal specifications, such as choreography and orchestration, but some activities such as data and process mediation (Cimpian and Mocan, 2005) have been implemented in WSMX. WSMO distinguishing characteristic is the explicit inclusion of goals and mediators in the conceptual model. In common with IRS-II, the WSMO approach builds on the UPML framework, taking advantage of the separation of tasks (Tasks/Goals) specifications from the service specifications (PSMs/Web Services). The orchestration description in WSMO has the same purpose as the process model in OWL-S, but it is still under development.

An assessment of the delivered results of IRS-II, OWL-S and WSMO approaches show that Semantic Web Services are far from mature. While they represent different developments converging to the same objective, they provide different reasoning support, which are based on different languages and ontology frameworks: IRS provides OCML as the language (Motta, 1999) based on the Ontolingua framework (Gruber, 1991); OWL –S is based on the OWL language under Description Logics; and WSMO is based on WSML which integrates a family of languages covering compatibilities with Description Logics, Rules and Logic Programming (Bruijn et al., 2006). Furthermore, none of the approaches described provides a complete solution according to the dimensions illustrated, but interestingly enough they show
complementary strengths. For example, IRS-II has strong user and application integration support while OWL-S provides a rich RDF-based service-ontology. WSMO has a comprehensive software architecture based on open-source, which covers SOA requirements.

Regarding mediation aspects, IRS-II contains the foundations for mediation that we will extend into IRS-III (see next chapter) to account for explicit mediation, choreography and orchestration based on the WSMO conceptual model. OWL-S does not model the mediator concept. Yet, mediation plays a key role in the approach (Paolucci et al., 2004). OWL-S considers that mediation is handled during discovery and decomposition by architectural components and that a mediation service is treated just as another web service. This assumption makes mediation very implementation dependent and not visible to the user. WSMX contains a data mediation component (Mocan et al., 2005) and a process mediation component (Cimpiam et al., 2005). The WSMX data mediation component can execute mapping rules generated at design time by a mapping tool. The WSMX process mediation component works at runtime on predefined types of mismatches between two choreography instances.

2.6 Mediation in other SWS Related Work

The deployment of Semantic Web Services will still rely on the further development and combination of Web Services and Semantic Web enabling technologies. There exist a number of other ongoing SWS projects which take into account mediation in their infrastructure as explained in the following.
SAWSDL\(^{23}\) (Semantic Annotation for WSDL) is a W3C standard specification for the annotation of WSDL based Web Services. The main characteristic of SAWSDL is the bottom-up approach to the semantic description of Web Services. SAWSDL defines extensions to the WSDL standard in order to link WSDL elements to ontology concepts and mapping functions, which are provided externally. An extension attribute, named `modelReference` is used to associate ontology concepts to XML Schema type definitions of service’s inputs and outputs as well as WSDL operations. Two extension attributes, named `liftingSchemaMapping` and `loweringSchemaMapping` are used to refer to mapping functions, which translate between ontology schema and XML Schema. A SWS framework using SAWSDL descriptions can disambiguate the descriptions of WSDL based Web Services during discovery and composition.

METEOR-S (http://lsdis.cs.uga.edu/projects/METEOR-S) is an approach for developing Web processes composed of SWS. This project uses the predecessor of SAWSDL, called WSDL-S for the semantic annotation of Web Services. The METEOR-S project comprises a set of integrated frameworks for annotation, execution, discovery and composition of SWS.

As part of the METEOR-S project above, an approach for data mediation is proposed in (Nagarajan et al., 2007). Data mediation between two Web Services is handled by lifting the service schemas to the semantic level using the mapping functions provided

\(^{23}\) http://www.w3c.org/2002/ws/sawsdl
by WSDL-S. Using schema transformation techniques, the source ontology schema is translated into the target ontology schema.

Another initiative for the development of SWS is the SWS Initiative (SWSI) (http://www.swsi.org), a consortium of academic and industrial partners, which investigates solutions for the main issues regarding the infrastructure for SWS, including architecture, and components for discovery and composition. In particular, a paper on mediation issues has been published under this initiative in (Burstein et al., 2003). It describes the issues related to mediation of messages (called translations) in the context of Semantic Web Services and agents. The examples focus on provided mapping knowledge (called articulations) carried by agents in terms of the concepts in the OWL and OWL-S, but there is no practical implementation available in the context of a SWS framework.

Within the WSMO approach (see Section 2.5.3), a component for data mediation (Mocan and Cimpian, 2007) and a component for process mediation (Cimpian and Mocan, 2005) have been implemented as part of WSMX execution environment. Data mediation is concerned with generating mapping rules at design time for mapping between instances of two domain ontologies. The mapping rules are stored and used at runtime by WSMX. The process mediator is a WSMX component which observes the behaviour of a service requester and a service provider according to their choreography descriptions and solves incompatibilities at runtime. The choreography description includes input and output modes (e.g. in, out, shared) and transition rules, which inform the order of messages. The process mediator can solve a few types of
incompatibilities such as split and concatenation of source outputs into target inputs.

We observe that WSMX is the only system that we know about that claims to perform process mediation at runtime; however, no results have been shown in the paper cited previously.

In (Vaculin and Sycara, 2007) an approach based on simulation techniques is presented for the automated mediation of processes modelled with OWL-S. The approach considers the control-flow structures of a requester process and a provider process as well as input concepts, output concepts, preconditions and post-conditions. The approach uses simulation of the execution path of the provider’s process model in order to find out mappings for the requester’s path. As a result, a mapping is constructed during the simulation, which is represented as a sequence of actions that the mediation component should execute during runtime mediation. The actions can be for example service invocations or data transformation requests. We observe that this approach, which is in fact used at design-time, has the benefit of considering data mismatches.

2.7 Data Mediation and Information Integration

Traditionally, the notion of data mediation has been associated with the automation of data integration in order to overcome heterogeneities in information systems in areas such as enterprise application integration (Wiederhold, 1992). A typical approach for mediation in this context usually involves the introduction of a common unified representation of data (a meta-model) to which heterogeneous information resources are mapped. The data mediator executes the provided mappings using the unified
representation when a retrieval request is invoked over the mapped resources. Through the mappings, the mediator performs tasks such as aggregation and disambiguation. Extended implementations of this approach consider ontologies as the common unifying representation (e.g. Sheth, 1998; Omelayenko, 2002) as well as Web Services as information resources (e.g. Liu, 2004). Thus, in this case, data mediation involves applying mappings between the common meta-level representation and the heterogeneous data formats.

In (Wiederhold, 1992, 1997) mediation is viewed in the context of information systems integration. The mediator is introduced as a middleware component in traditional client-server architecture and is designed to execute intermediate tasks that should neither be part of the client nor the server. The mediation tasks, also called mediating services, carried out by a mediator are: selection of high quality data, matching of source data; creating fused data objects, summarizing and abstracting from source data. In this approach, user queries against multiple heterogeneous data sources are communicated via the mediator (middleware) which is responsible for query rewriting, dispatching query fragments to individual sources, and assembling individual query results into a composite query response. Within this framework, source heterogeneity is managed by source wrappers which export a common data model view of the data at each source, along with the information about the source schema and metadata, and a description of the supported queries. This is a reference work that introduces the mediator as a key component for handling data heterogeneity within the information integration context. The strongest argument of this work is that of a clear separation of concerns between mediation services and back-end services.
2.8 Process Mediation and Web Service Composition

Composition or orchestration of Web Services is one of the predominant activities when building service-based business applications, as demonstrated by the number of existing composition languages available from industry (Wohed et al., 2003). These languages enable the composition of Web Services into business processes.

Composition can thus be defined as the task of finding suitable Web Services for fulfilling business process workflow activities. We refer to Process Mediation in this context when the composite process is in fact controlling the invocation of Web Services from partner processes so that they can interact. Hence, the modelling of the mediator process is based on the combination of control-flow, data-flow and atomic activities of two or more processes being mediated.

In this section we review related work on Web Service composition, which falls on a variety of categories: static (hand-written) composition based on syntactic workflow standards (case of BPEL); dynamic (automatic) composition based on syntactic workflow formalisms (the case of Petri-Nets for instance); dynamic composition based on semantic workflows (case of AI based approaches); static and semi-automated composition based on semantic workflow standards (case of OWL-S process model for instance); and semi-automated composition based on syntactic workflow standards and SWS (case of SAWSDL for instance).

There is substantial research work aiming at automatic composition, that is, providing techniques to support the automatic generation of the process mediator (workflow)
based on classical formalisms such as, Petri-nets (Hamadi and Benatallah, 2003) or Process Algebra (Salaun and Shaerf, 2006). Although these approaches succeed in producing a composition or workflow, they are usually limited in the types of control flow constructs they can support (usually only sequences) and do not take into consideration data heterogeneities. That is, they typically assume that a control flow can be generated based on the exact matching between request (input) and response (output) formats of the activities from the participating processes.

Semantic-based approaches to automatic composition based on Planning can be found in (McIlraith and Son, 2002) and (Traverso, P. and Pistore, M, 2004). For example, the work in (Traverso, P. and Pistore, M, 2004) claims to provide the automated composition of Semantic Web Services, in which a plan (the mediator or composition) can be synthesized given a set of Web Services. In this case they take the view of Web Services as abstract processes in which only the interactions (request and response messages) are visible. The plan is produced according to a complex goal expressed at the knowledge-level (a specific logic-based language called EAGLE). The produced plan is a simple workflow (sequence of Web Services) with the potential to be translated into an executable format (BPEL4WS). In this case, the success of obtaining a plan is heavily dependent on the specific formulation of a complex goal.

In (Szomszor et al., 2006) the authors propose a mediation approach for integrating Web Services into workflows in the bioinformatics domain. The main contribution from this work is a mapping language based on XSLT that transforms data from
different XML Schema formats (from heterogeneous services) to OWL XML instances (OWL-XI) and conversely. The integration approach presented uses OWL as the common language to which other logical formats can be mapped to. Services with semantically equivalent inputs and outputs (via the mappings) can then be combined. The mappings (manually provided) are used at runtime to translate between the syntactic and semantic levels. We observe that with this approach, resolving the data mismatches takes the same effort as if only syntactic descriptions were used (as part of the workflow language). Also, it is not straightforward to inspect which services are using which concepts. So, it is hard to maintain the system when there are changes.

In (Mandell and McIlraith, 2003) the authors argue that the syntactic approach provided by BPEL4WS have shortcomings that limit its ability to provide seamless interoperability. Thus, they propose the use of semantic-based technologies to support automated service discovery, customization and semantic translation for BPEL4WS based processes. Their service discovery and customization approaches use OWL-S annotations as the basis for dynamically querying and selecting services from partners. In addition, they provide a semantic translation algorithm to resolve data mismatches between service partners, by which a chain of given translation services is built automatically at runtime to produce the required input of a service. They assume that translation services are programs encoding translational axioms and exposed as Web Services. The mediation component is a software component that interacts with the BPEL4WS engine during execution to provide the necessary translations. We observe that this is one of a few early approaches providing semantic mediation of
SWS using OWL-S. Similar to the previous paper, it is not straightforward to inspect which services are using which mediation services. So, it is hard to maintain the system when there are changes.

In (Aslam et al., 2006) a tool for mapping BPEL4WS to OWL-S is provided in order to provide semantics to business process models. In this approach primitive activities (invoke, send, reply) and structured activities (e.g. sequence, while) as well as data flow can be mapped from BPEL4WS to OWL-S, but there are some problems in mapping conditions, assignments, fault handlers and synchronization constructs from BPEL4WS. In addition, input and output parameters in the resulting ontology need to be annotated with domain ontologies by the user.

Another example of formalism for mediation is the work in (Altenhofen et al., 2005) which provides an abstract specification for Mediators (Virtual Providers) based on Abstract State Machines (ASM). The specification is used to design business processes. The mediators represent workflow components (e.g. Send and Receive) which can be composed into processes.

### 2.9 Mediation and Decentralized Data Management

Work in the context of peer-to-peer data management breaks from the traditional approach by (Wiederhold, 1992) and opposes to the idea of using a centralized data schema for integrating heterogeneous data sources. Instead, these systems use mappings stored elsewhere for matchmaking (querying). Thus, the focus is on the decentralized coordination and management of peer systems aiming at data integration. The Piazza system (Halevy et al, 2004) is a good example of a
decentralized data management system which includes a formalized schema mediation mechanism. This work is extended to the Semantic Web by using OWL ontologies.

The work of (Bouquet et al., 2004) is also on the coordination of peers, focusing on a method for discovering semantic relations between peers instead of using a shared model.

### 2.10 Mediation and Agent Based Systems

There is also substantial work on Mediation in the area of Agent Based Systems with respect to the interoperation of heterogeneous agents, which we select and analyse in this section so that we can inform our approach. However, while the research problems in the Agents area might be similar to the ones in the SWS area, agents as software artefacts are rather different in nature from SWS. Basically, the typical agent features of autonomy, social ability, reactivity and pro-activeness (Wooldridge and Jennings, 1995) do not apply to SWS. However, SWS can be seen as a way of implementing agents as services. As explained in Section 2.2, Web Services provide the building blocks for SWS and as such follow a much simpler (non-cognitive) communication protocol, interface description and semantic/reasoning formalism than agents.

In (Diggelen et al., 2005) a decentralized approach to agent communication is proposed, where agents teach unknown concepts to each other, dynamically creating a communication vocabulary. The authors correctly highlight that a decentralized
approach to communication is necessary since a common ontology is disadvantageous for the problem solving capabilities of the agents as different tasks typically require different ontologies (see also (Bylander and Chandrasekaran, 1987)). We note that this approach circumvents mediation by counting on the agent’s ability to learn a vocabulary. However, SWS are not ascribed with this ability, rather they respond to fixed-format messages.

In (Laera et al, 2006) an argumentation framework is proposed to enable agents to communicate by reaching agreement over ontology alignments. The framework allows agents to express their preferred choices over candidate alignments. The implementation of the proposed argumentation formalism uses OWL to express alignments. We agree with the authors that alignments (correspondences or mappings) between heterogeneous agent’s ontologies are necessary in order for agents to interact. However, we disagree when the authors say that their framework can equally be applied to Semantic Web Services since they are not endowed with the ability to express interests and preferences about the terminology they use. We note again (as in the work discussed before) that this approach circumvents mediation by counting on the agent’s ability to negotiate.

The authors in (Trojanhn et al., 2008) propose two agent-based techniques for the problem of composite ontology mapping. Basically, they use a negotiation model and an argumentation formalism in order to decide on matching algorithms provided by different agents. The goal is to arrive with a better matching result between ontology elements through agent cooperation than would be possible using an individual agent.
The negotiation technique presented is a mediation approach itself, where a mediator agent asks and receives mapping proposals from the other agents. We note that this mediator is just a program to evaluate proposals from the other agents; there is no query or mapping resolution. The paper also makes a survey and a classification of different ontology matching approaches, taking into account lexical, structural and semantic based techniques. We observe that in the context of SWS, these matching algorithms can be used for discovery of SWS, which is the phase before mediation (where mismatches are resolved). Also, constraint-based matching techniques are more applicable to SWS discovery because service descriptions represent tasks that might have restrictions. Mappings must still be supplied for bridging between combined SWS that do not match.

### 2.11 Summary of Research Problems

From the reviewed work described in the last sections, we derive a list of basic research problems on mediation, which we will undertake as follows.

**Mapping data schema to ontologies** – As seen in the previous section, mapping data schemas (e.g. database schema or xml schema) to ontologies allows for an agreement on the semantics of heterogeneous syntactic data formats. Through these mappings one can provide semantic interoperability, since different data formats or terms may refer to the same concept, or the same data format or term may refer to different concepts. Data mediation takes place when a query is performed over the ontology and the mappings are applied so that the applicable resources can be retrieved. We refer to this type of data mediation as *lifting* when going from the syntactic level to
the semantic level, and lowering when going from the semantic level to the syntactic level. There is a general problem of scalability with this approach when used alone since all resources must agree with the same ontology for interoperability. For example, when Web Service A is annotated with Ontology A, it can only interoperate with Web Service B if B itself is also annotated with Ontology A.

**Mapping ontologies to ontologies** – Extending the problem above, the data heterogeneity problem is escalated to the semantic level when resources are annotated using different ontologies. As mentioned in section 2.3, there is a whole area of research on ontology mapping for dealing with the problem of aligning ontologies. Regarding Semantic Web Services, the important issue is how to use the mappings generated from the process of alignment for different activities.

**Separation of Concerns** - As proposed in some of the work reviewed previously, separating mediation issues from the main components (i.e. client or server) can improve performance and promote reuse and scalability. We follow on this idea by separating mediation aspects at the semantic level from the ontology models representing capabilities of the request (Goal) and of the provided service (Web Service) and using mediators to map between these models.

**Support for Discovery and Composition** - As mentioned in the previous section, mediation can take place as part of discovery (resource querying) and composition. The problem here is that business applications cannot depend on mappings between services that are not completely accurate. On the other hand, we observe from the literature review above that typical approaches for automatic composition and
discovery tend to overlook data mismatches. Given these problems when considering business requirements, we will favour a declarative approach to composition and mediation.

**Decentralization.** We will mainly follow on a decentralized approach to mediation, which is important for scalability in an open and distributed environments such as the Web. Instead of counting on one centralized ontology into which separate resources are mapped so that they can interoperate; we follow on the idea of annotating services independently and providing mappings between the ontologies.
Chapter 3  A Framework for Semantic Web Services

In this chapter we present our first contribution, which is based on the partial design and implementation of the IRS-III framework (also published in Cabral et al., 2006). First, we formulate a consolidated characterization for Semantic Web Services following on the comparison of existing technologies and approaches given in the previous chapter, by defining a conceptual framework for Semantic Web Services according to a service ontology, usage activities and execution components. Then, we provide a description of the IRS-III approach that we have implemented accordingly by extending IRS-II (Chapter 2, Section 2.5.1). Our goal is to introduce the Semantic Web Service infrastructure which we have contributed to as a whole, before going into more details of the mediation framework (described in the next chapter).

More specifically, our contribution to the IRS-III framework consists of the definition and implementation of the mediation framework (described in detail in the next chapter), the definition of the orchestration and choreography primitives; the development of the IRS-III browser and java API; and the development of the java Publisher and the WSDL Publisher. The work on IRS-III started in parallel with the foundation of the WSMO Working group (http://www.wsmo.org) and as part of the DIP project (http://dip.semanticweb.org/).

3.1 Problems with Existing SWS Approaches

One major deficiency in the area of Semantic Web Services is the lack of a consolidated view characterizing the many aspects of SWS, including semantic
descriptions, usage and implementation. The first contribution of this chapter is the formulation of a SWS conceptual framework, which fulfils this need.

In addition, due to the immaturity of the research area, existing approaches tend to focus on specific issues (e.g. discovery or composition); therefore, there is a major lack of infrastructure, including tools, development and execution environments, which can be utilised for the description and the execution of Semantic Web Services as well as the implementation of applications. Thus, as our next contribution, we implemented a number of IRS-III components in order to fulfil the need for a comprehensive Semantic Web Service infrastructure.

3.2 A conceptual framework for SWS

We characterize a Semantic Web Service (SWS) infrastructure along three orthogonal dimensions: service ontologies, usage activities and architecture components. The service ontologies constitute the conceptual models describing what a service does and how to use it, also used for reasoning during the whole life cycle of Semantic Web Services. Usage activities define the application requirements that a framework for Semantic Web Services ought to support. The architecture components are part of the execution environment needed to accomplish these activities. We describe these dimensions in detail in the remainder of this section.

3.2.1 SWS Service Ontology

The service ontology is a set of conceptual models that enables the description of the capabilities of a Web Service and the restrictions applied to its use. It essentially
integrates at the knowledge-level the information necessary for understanding what
the service does, how it does it and how it can be used. The service description is
usually used before the service is invoked in activities such as discovery (e.g.
matchmaking based on concepts and non-functional properties) and analysis of
service behaviour. For invocation, input values must be given and mappings must be
available. Below we describe the main elements that may be included in the service
ontology.

**Functional Capabilities** – represent the provider perspective of what the service does
in terms of inputs, output, pre-conditions and post-conditions. Pre-conditions and
post-conditions are expressed by logical expressions that constrain the state or the
type of inputs and outputs; or in some cases may constrain the information given by
the related domain ontology.

**Goal-Related Information** – represents the user perspective of the required functional
capabilities. It can the same functional capabilities as described above;

**Choreography** – represents the interaction logic between a client and a service, which
can be expressed in terms of message patterns.

**Orchestration** – represents the interaction logic of sub-services, which may be
expressed with a workflow model. It may include workflow constructs (e.g. activities,
events, gateways) or rules. It can also be referred to as the composition of services.

**Non-Functional Properties** - can range from information about the provider such as
company and address, to information about the service such as category, cost and
quality of service, to execution requirements such as scalability, security or robustness.

*Meditators* – provides a container for mappings or mediation services between Semantic Web Service elements for solving mismatches.

### 3.2.2 SWS Usage activities

Under the usage perspective, SWS can be seen as the building blocks of a business application. The activities required for developing an application using SWS extend the ones for Web services (described in Chapter 2, Section 2.2), and includes: publishing, discovery, selection, composition, invocation, mediation, service deployment, ontology reasoning and invocation, as described next.

**Publishing** - The publishing or advertisement of SWS allows software agents or applications to discover services based on goals or functional capabilities. A semantic registry can be used for registering instances of the service ontology for individual services. The service ontology distinguishes between information which is used for matching during discovery and that used during service invocation. In addition, domain knowledge may also be linked to the service ontology.

**Discovery** - The discovery of services consists of a semantic matching between the description of a service request and the descriptions of published services. The matching criteria can be based on some pre-defined logical relation such as exact matching, intersection, inclusion or subsumption. For example, an input of type
Professor can be said to match an input of type Academic by subsumption. The matching can also be done between services and a goal to be achieved.

Selection - A selection of services is required for choosing amongst discovered services matching the request. A ranking criterion based on non-functional attributes such as cost or quality may be used for selecting one service. In general, a SWS broker would check whether the pre-conditions of a service are satisfied against input values and prove that the service post-conditions and effects imply goal accomplishment.

Composition - An orchestration or process model for SWS may be defined in the service ontology to express through control constructs how external services can be combined to achieve a desired functionality.

Mediation - Mediation involves handling the mismatches between a service requester and one or more service providers. Data mediation solves mismatches between ontologies through mapping, merging or alignment. Process mediation solves behavioral mismatches between two interacting Web Services.

Deployment - Web services are expected to be deployed on the Web by various providers who must also provide a semantic description for them during publishing. However, one facility that can be provided by the SWS infrastructure is the deployment of code as Web Services.

Ontology Reasoning - Ontology reasoning is a cornerstone activity for SWS. It is essentially based on the language adopted and its expressivity power. The language is
the basis for service modelling, querying and mapping. Reasoning is done not only over domain instances and logical expressions, but also over the service descriptions.

**Invocation** - The automatic invocation of SWS involves a number of steps, starting with the provision of required input values by the requester. First, the service and domain ontologies associated with the service must be loaded. Second, the input values must be lifted and validated against the ontology types and the preconditions satisfied. Third, if a service is composite, its workflow must be executed. Finally, atomic services are called through the grounding provided. Also, monitoring the status of the invocation process and notifying the requester is important.

### 3.2.3 SWS Architecture Components

A SWS infrastructure should realize the activities described above with underlying security and trust mechanisms. The architecture components derived from the discussion above would include: a registry, a reasoner, a matchmaker, a decomposer, a mediator and an invoker. These components are illustrative of the required roles in the SWS architecture for the discussion here as they can be named differently and have a complexity of their own in different approaches.

**Reasoner** - The reasoner or inference engine is used during all activities for interpreting semantic descriptions and performing queries over ontologies. As with all logical systems, deductions on the ontology instances can be derived according to the language used.
**Registry** - The registry provides the mechanisms for publishing and searching services. A semantic registry may allow a search of services through logical queries. It may provide functionalities for creating, editing and browsing service descriptions as well.

**Matchmaker** - The matchmaker component is used during discovery and selection for finding services that match a specific request.

**Decomposer** - The decomposer is the component required for executing the process model of composed services.

**Mediator** – Different types of mediator components can handle mismatches during the discovery, selection, composition or invocation of Semantic Web Services by reasoning over the instantiated service ontologies.

**Invoker** - The invoker component will invoke the actual deployed service after all the semantic activities have been resolved.

### 3.3 Implementing the SWS Framework

In this section we present IRS-III, which implements the SWS conceptual framework described in the previous section. The development of IRS-III started in January 2004 within the context of the DIP project (http://dip.semanticweb.org) as an extension of IRS-II (Chapter 2, Section 2.5.1). With IRS-III, we aimed at providing an open execution environment for Semantic Web Services compatible with the WSMO standard. As shown in chapter 2, IRS-II and WSMO inherited both from the UPML framework, thus facilitating our extension from the IRS-II’ Task-PSM conceptual
model to the WSMO conceptual model in IRS-III. Mainly, we converted Task models into Goal models; and PSM models into Web Service models. Then, all the other properties (e.g. non-functional properties) were added or adapted. The major new additions of IRS-III are the design of mediation, choreography and orchestration, which has no prescribed implementation in the WSMO specification. Regarding Mediation, it is worth noting that we take the notion of Bridge from the previous IRS implementation (Crubezy et al., 2003) and extend this notion into a mediation model as first class citizen in IRS-III.

3.3.1 The IRS-III approach

In IRS-III we adopt the WSMO conceptual model (WSMO Working Group, 2004) and implement the top-level entities as knowledge-based models which can be associated with deployed Web Services. The four top level elements of the framework are:

- **Ontologies.** Provide the foundation for semantically describing data in order to achieve semantic interoperability and are used by the three other elements;

- **Goal.** Define the task that a service requester expects a web service to fulfil. In this sense they express the client request;

- **Web Service.** Represent the capability of an existing deployed Web Service. The description also outlines how Web Services communicate (choreography) and how they are composed (orchestration);
• **Mediator.** Describe connections between the components above and represent the type of conceptual mismatches that can occur. In particular, four types of mediators are provided (adapted from WSMO): OO-mediators link and map between heterogeneous ontologies; WG-mediators connect web services to goals; GG-mediators link different goals; WW-mediators link web services to web services.

The following describes the main application development activities supported by IRS-III when building Semantic Web Services:

• **Using Domain Ontologies.** The concepts involved in the application scenario which are used to describe client requests and Web Service capability are provided in domain ontologies.

• **Describing Client Requests as Goals.** The request for a service can be expressed from a business viewpoint and represented as a goal.

• **Semantically Describing Deployed Web Services.** The concepts defined in domain ontologies can be used in a Web service description to represent the types of inputs and outputs of services and in logical expressions for expressing applied restrictions. This description can also include many other aspects such as orchestration and choreography.

• **Resolving Conceptual Mismatches.** Mediator descriptions can be used to declare a mediation service or mapping rules that will be used to provide alignment between goals, web services and domain ontologies.
• **Publishing Semantically Described Web Services.** Once a semantic
description has been created for a deployed Web Service as above, it can be
registered into the IRS-III repository for goal-based invocation.

The IRS-III tooling consists of a Java API and a browser/editor which support
developers in building applications out of Semantic Web Services. The IRS-III
browser provides an easy to use graphical interface to support the creation of semantic
descriptions, to publish deployed Web Services against these descriptions and to
invoke Web Services. The IRS-III Java API provides a data model for our framework
and remote access to the IRS-III server. We have also developed a plug-in for WSMO
Studio (Dimitrov et al., 2005) for interoperability purposes, by aligning the IRS-III
and WSMO4J (http://wsmo4j.sourceforge.net) APIs.

### 3.3.2 IRS-III Design Principles

The IRS-III approach is based on the following set of design principles:

• **Service Ontologies as Knowledge models** – Within IRS-III, service ontologies
are provided as knowledge models representing the SWS top-level elements.
These knowledge models are semantically linked and are represented using
our ontology representation language OCML (Motta, 1999).

• **Top-down Approach** – We advocate a top-down approach to the annotation of
Web Services, in which we first create semantic descriptions (instances of the
IRS-III knowledge models) and then ground these descriptions to an actual
Web Service syntactic interface description (e.g. WSDL). In this way, it is
possible to have more than one semantic description (or view) over the same implemented service (code). In addition, the same semantic description can be grounded to different service interface description standards (e.g. WSDL or REST).

- **Reasoning is Ubiquitous** – IRS-III is an operational semantic environment. It relies on the underlying language OCML and its proof system (reasoner). Most activities performed by IRS-III during runtime including mediation, selection and orchestration are implemented in OCML. The OCML reasoner is used for performing ontological queries over the service descriptions and domain instances.

- **Reasoning is Ubiquitous** – The IRS-III execution environment relies on the operational power of the underlying language OCML and its proof system. Reasoning takes part in all Semantic Web Service activities. OCML is used for implementing ontological queries over the service descriptions and domain instances.

- **Goal-based Invocation** – IRS-III supports a goal-centric invocation mechanism. A client application simply asks for a Goal to be achieved and IRS-III selects an appropriate Web Service, invoking the associated deployed code.

- **Goal-based Decomposition** – In IRS-III a Web Service is either atomic (executable) or composed of other Goals. A composite Web Service is
expressed in terms of a workflow based on Goals, which follows on the previous design principle for invocation.

- **Explicit Mediation Description** – IRS-III uses the Mediator description for two purposes. First, it serves as a placeholder for mappings or a mediation service. Second, mediators can represent different types of mediation, which can be related to different mediation activities.

- **Use of Publishers** – IRS-III delegates the execution of code associated with a Web Service to the Publisher (see architecture description below). That facilitates the execution of code from different platforms, such as Java, Lisp or platforms supporting WSDL.

- **IRS-III as a broker** – IRS-III mediates between client requests and service providers whenever a Semantic Web Service is invoked.

### 3.3.3 The IRS-III Service Ontology

The IRS-III service ontology follows on the conceptual framework presented previously (Section 3.2). It extends the IRS-II knowledge models with the capabilities specified in the WSMO conceptual model as follows:

- **Non-Functional Properties** – These properties are associated with the main top elements and can range from information about the provider such as organisation, to information about the service such as category, cost or trust, to execution requirements such as scalability, security or robustness.
• **Goal-related Information** – A goal represents the client perspective of the required functional capabilities. It includes a description of the requested web service capability.

• **Web Service Functional Capabilities** – Represent the provider perspective of what the service does in terms of inputs, output, pre-conditions and post-conditions. Pre-conditions and post-conditions are expressed by logical expressions that constrain the state or the type of inputs and outputs.

• **Choreography** – The choreography of a Web Service specifies how to communicate with that Web Service.

• **Grounding** – The grounding is associated with the Web Service choreography and describes how the semantic descriptions are mapped to a syntactic specification such as WSDL.

• **Orchestration** – The orchestration specifies how to provide functionality in terms of the composition of one or more Web Services.

• **Mediators** – connect top elements and specify how to resolve types of mismatches.

The IRS-III service ontology differs from the WSMO conceptual model in the following ways:
• **Explicit Input and Output Role Declaration** – IRS-III requires that Goals and Web Services have explicit input and output declarations, which include a name and a semantic type.

• **Web Services are Linked to Goals via Mediators** – In IRS-III, the basic mechanism used to link a Web Service to a Goal is to put a mediator between them. All linked Web Services are candidates for selection; but pre-condition and assumption expressions are used to further refine the applicability of the Web Service.

• **Mediators Provide Data-Flow between Sub-Goals** – In IRS-III, Mediators can be used to link sub-goals within an orchestration and so they also provide dataflow between the sub-goals.

• **Web Services can Inherit from Goals** - Web services which are linked to Goals inherit the Goal’s input and output roles. This means that input declarations within a Web Service are not mandatory and can be used to either add extra input roles or to change an input role type.

• **Client Choreography** – The provider of a Web Service must describe the choreography from the viewpoint of the client. This means IRS-III can interpret the choreography in order to communicate with the deployed Web Service.
• **Mediation Services are Goals** – A Mediator declares the Mediation Service as a Goal, which can simply be invoked. The deployed Web service actually performs the necessary data transformation.

### 3.3.4 The IRS-III Architecture

In this section we describe the IRS-III architecture components, which handle the service ontology described above. IRS-III extends the IRS-II architecture (Chapter 2, Section 2.5.1). IRS-III is composed of the IRS-III Server, the Publishing platforms and Clients which communicate through the SOAP protocol, as shown in Figure 3-1.

![Figure 3-1 The IRS-III Architecture](image)

The IRS-III Server handles ontology management and the execution of knowledge models. The Server also receives SOAP requests (through the API) from client applications for creating and editing descriptions of *Goals, Web Services* and *Mediators* as well as requests for Goal-based invocation. At the lowest level the IRS-
III Server uses an HTTP server written in Lisp, which has been extended with a SOAP handler.

The publishing platforms allow providers of services to attach semantic descriptions to their deployed services and provide handlers to invoke services in a specific language or platform (Web Services WSDL, Lisp code, Java code, and Web applications). When a Web Service is published in IRS-III the information about the publishing platform (URL) is also associated with the Web Service description in order to invoke the Web Service. 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.

The main components of the IRS-III are explained in the following:

- **SWS Library** – At the core of the IRS-III server is the SWS library where the semantic descriptions are stored using our representation language OCML (Motta, 1999). The library is structured into knowledge models for goals, web services and mediators. Domain ontologies and knowledge bases (instances) are also available from the library.

- **Choreography Interpreter** – This component interprets the choreography description, which contains the grounding and guarded transitions for handling the communication with the deployed Web Service (more details are given in the section below).
• **Orchestration Interpreter** – This component interprets the orchestration description used in composite services (more details are given in the section below).

• **Mediation Handler** – Selection, composition and invocation are each supported by a specific mediation component within the mediation handler. These activities may involve executing a mediation service or mapping rules declared in a mediator description. More details are given in the section below.

• **Invoker** – The invoker component of the server communicates with the publishing platform, sending the inputs from the client and bringing the result back to the client.

The following sections give more details of the reasoning engine and how choreography, orchestration and mediation of Semantic Web Services have been implemented in IRS-III.

**IRS-III Choreography**

The IRS-III choreography model describes how IRS-III interacts (exchange messages) with a single deployed Web Service (client choreography). The Choreography model, expressed in OCML, represents a set of forward-chaining rules and a grounding declaration. The action part of the rules are based on communication primitives which are executed when the associated conditions (facts) are satisfied (matched). The grounding declares the operations and messages (input and output) involved in the invocation of the SWS as well as the mappings between the semantic and syntactic
level. More specifically, each operation's input and output is associated with a lifting or lowering function. The *grounding* also describes information about the corresponding publishing platform. This approach allows the functionality of a Web Service to be realized by calling one or more declared operations. The communication primitives are described below.

- **init-choreography** – The initial assertion made by IRS-III when the state of the choreography is initialized. IRS-III obtains the input values of operations from the goal invocation request.

- **send-message** - Calls a specific operation in the associated Web service.

- **received-message** - Contains the result of a successful send-message for a specific operation.

- **received-error** - If an operation generates an error then this primitive is used including the error message and the name of the operation causing it.

- **end-choreography** - Stops the choreography. No other rule will be executed.

More details about the formalization of the IRS-III choreography, can be found in (Domingue et al., 2006).

**IRS-III Orchestration**

In IRS-III the orchestration is used to describe a composed Web Service. The orchestration model, expressed in OCML, represents a workflow model. The distinguishing characteristic of this model is that the basic unit within the composition
is a goal. Thus, the model provides control and data flow constructs over a set of
goals. Further, dataflow and solving mismatches between goals are supported by
mediators. The set of control flow primitives which have been implemented so far in
IRS-III are listed below.

- **orch-sequence** – Contains the list of goals to be invoked sequentially. A GG-
  mediator can optionally be declared with the goals, in which case the output of
  the source goal is transformed by the mediation service (if there is one) and
  used as input of the target goal.

- **orch-if** – Contains a condition and a body with one or more workflow
  primitives. The body part is executed if the declared condition is true.

- **orch-repeat** – Contains a condition and a body with one or more workflow
  primitives. The body part is repeated until the declared condition is false.

- **orch-get-goal-value** - Returns the result of the last invocation of the declared
  goal (used for example as part of a condition).

- **orch-return** – Returns the result of the current goal execution.

Additional work is under specification in order to refine and provide further
capabilities.

**IRS-III Mediation**

At the semantic level, IRS-III represents four basic types of conceptual mismatches
that can occur when using Semantic Web Services. These types correspond to the
WSMO models of OO-mediator, WG-mediator, GG-mediator and WW-mediator as described in Chapter 2, Section 2.5.3. In general there will be mismatches between the Goal requests and available Web Services and between the Goals themselves. The IRS-III mediation handler components are responsible for resolving the conceptual mismatches which may occur by reasoning over the given goal, web service and mediator descriptions. The mediation handler interprets each type of mediator accordingly during selection, invocation and orchestration. More details of mediation in IRS-III will be given in the next chapter and can also be found in (Cabral and Domingue, 2005).

Basically, the Mediator model declares a source component, a target component and either a Mediation Service or Mapping Rules. Hence, the Mediator provides a semantic link between the source component and the target component, which enables Mediation Services or Mapping Rules to solve mismatches between components. In this model, the mediation service is just another goal. As an example, a Mediation Service of a WG-mediator transforms input values coming from the source Goal into an input value used by the target Web Service. Mapping Rules are used between two ontologies (source and target components). These mappings only concern to the concepts used during invocation.

### 3.4 Overview of the Languages OCML and WSML

In this thesis we will use both OCML and WSML as languages to model Semantic Web Services as both implement the WSMO meta-model that we adopt in our work. More specifically, we use OCML to implement IRS-III as seen in this chapter and
later in Chapter 4 to implement the mediation framework. We use WSML for building the Business Process Modelling Ontology (Chapter 5) and a translator. In particular, using the two languages allowed us to abstract the mediation aspects from the language and experiment with slightly different mapping mechanisms (see also the discussion in Chapter 5).

In this section we present a brief overview of these languages, taken from the documentation available, and compare them with respect to their usability in our work.

### 3.4.1 OCML – Operational Conceptual Modelling Language

OCML (Operational Conceptual Modelling Language) (Motta, 1999) is an operational language for ontology and knowledge modelling. That is, the language is integrated with an interpreter and a proof system (reasoner engine) on top of a Lisp-based environment. OCML combines a frame system with an integrated forward and backward chaining rule system. OCML constituent constructs include classes, instances, relations, functions, procedures and rules, following on the Ontolingua formalism (Gruber, 1993). In OCML, procedures and functions can be attached to Lisp code. Classes are unary relations and class attributes are binary relations. Moreover, relations in OCML can be defined as forward or backward chaining rules. The operational semantics of the forward chaining system are equivalent to OPS5 (Forgy, 1981) and the backward chaining rule system has equivalent operational semantics to Prolog (Clocksin and Mellish, 1984). OCML has been used in a wide variety of projects covering knowledge management (Domingue and Motta, 2000).
OCML is the core component of IRS-III (as can be seen in Figure 3-1). All the major server components use the OCML reasoner; and the IRS-III service ontology is defined with the OCML language.

Details of OCML syntax, the semantics of the interpreter and the OCML proof system are given in (Motta, 1999). Below we present the main elements of a Knowledge Model or ontology in OCML (see examples in Listing 3-1):

- **Classes** - Classes are defined by means of a Lisp macro, `def-class`, which takes as arguments the name of the class, a list (possibly empty) of superclasses, optional documentation, and a list of slot (attribute) specifications. The semantics of `class` is the same as unary relations. The semantics of `slots` are the same as binary relations with respect to the class, for the purpose of querying.

- **Class Instances** – Class instances are simply members of a class. An instance is defined by means of `def-instance`, which takes as arguments the name of the instance, the parent of the instance (i.e. the most specific class the instance belongs to), optional documentation and a number of slot-value pairs.

- **Relations** - Relations allow the OCML user to define labelled n-ary relationships between OCML entities. Relations are defined by means of a Lisp macro, `def-relation`, which takes as arguments the name of a relation, its argument schema (a list of variables), optional documentation and a number of relation options. Relation options provide formal specifications with different semantics, which are used to find out whether or not a relation holds for some arguments. Relations also have an operational role in OCML. Some of the
available relation options are :iff-def, :sufficient, :constraint, :prove-by and :lisp-fun.

- **Rules** – These are backward rules and forward rules. Backward rules play a specification role and provide a dynamic mechanism to associate values between relations. They have the same semantics as relations with options :iff-def. When carrying out a proof by means of a backward rule, the OCML interpreter will try to prove whether some tuple is an instance of the relevant relation by firing the clauses in the order in which these are listed in the rule definition. As in Prolog, depth-first search with chronological backtracking is used to control the proof process. Forward rules provide a mechanism for changing the state of the running system (e.g. asserting or deleting facts); or causing external actions (e.g. printing information).

- **Functions** - Functions are defined by means of a Lisp macro, def-function. This takes as argument the name of a function, its argument list, an optional variable indicating the output argument, optional documentation and zero or more function specification options. The :body option for instance allows the attachment of a functional term which is evaluated in OCML’s environment.

- **Procedures** - Procedures define actions or sequences of actions which cannot be characterized as functions between input and output arguments.

- **Logical expressions** - The simplest kind of logical expression is a relation expression, which has the form (rel {fun-term} *), where rel is the name of a relation and fun-term is a functional term. More complex expressions can be
constructed by using the logical operators *and*, *or*, *not*, *=>*, *<=>* and quantifiers *forall* and *exists*.

**Listing 3-1 Examples of OCML Syntax and Elements**

```ocml
(def-class organisation (legal-agent)
  (has-name :type string)
  (has-size :cardinality 1 :type organisation-size))

(def-instance kmi (organisation)
  (has-name "Knowledge Media Institute")
  (has-size medium))

(def-relation enumerated-set (?x)
  "A set represented as (set-of e1 e2... en), where no ei is repeated"
  :iff-def (and (= ?x (set-of ?elements))
             (not (exists ?el
                       (and (member ?el ?elements)
                            (member ?el (removel ?el ?elements)))))))

(def-relation involved-in-project (?x ?project)
  "This relation associates people to projects"
  :iff-def (or (has-project-member Pproject ?x)
               (has-project-leader Pproject ?x)))

(def-rule infer-involved-in-project
  ((involved-in-project ?x ?project)
   if (has-project-leader Pproject ?x)))

(def-relation-instances
  (involved-in-project liliana-cabral dip)
  (involved-in-project john-domingue dip))

(def-function rest (?l)
  "Returns the elements of a list but the first one. If the list is empty, then NIL is returned"
  :constraint (list ?l)
  :body (if (= ?l (?a . ?b))
          ?b
          nil))
```

3.4.2 WSML – Web Service Modelling Language

WSML (Web Service Modeling Language)\(^{24}\) is a family of formal ontology languages that are specifically aimed at providing means to formally describe all the elements defined in WSMO. WSML semantics is based on Description Logics, Logic Programming and First-Order Logic, with influences from F-Logic and frame-based...

\(^{24}\) [http://www.wsmo.org/TR/d28/d28.3/v0.2/20070416/d28.3v0.2_20070416.pdf](http://www.wsmo.org/TR/d28/d28.3/v0.2/20070416/d28.3v0.2_20070416.pdf)
representation systems. Conforming to the different influences, there exist five variants of WSML: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full. Currently, the WSML2Reasoner framework has been implemented and tested using various reasoning engines that support all WSML variants with the exception of WSML-Full. In particular, IRIS (Integrated Rule Inference System), is an extensible reasoning engine for expressive rule-based languages. IRIS supports Datalog extended with stratified and well-founded negation, function symbols, unsafe-rules, XML schema data-types and an extensible set built-in predicates.

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; these are the elements of the WSMO conceptual model. Logical expressions are used to refine these definitions using a logical language.

The conceptual syntax for WSML has a frame-like style. The 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, which consists of such things as WSML variant identification, namespace references, non-functional properties (annotations), import of ontologies, references to mediators used and the type of the specification. This meta-information

---

25 http://tools.sti-innsbruck.at/wsmI2reasoner/
26 http://iris-reasoner.org/
27 http://www.wsmo.org/TR/dl6/dl6.l/v0.21/
block is strictly ordered. The second part of the specification, consisting 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., is not ordered. WSML allows creation of axioms in order to refine the definition already given in the conceptual syntax, i.e., the sub-concept and attribute definitions. Listing 3-2 shows an example of the WSML syntax and Ontology elements.

**Listing 3-2 Example of WSML Syntax and Ontology Elements**

```xml
namespace "http://www.example.org/ontologies/example#",
  dc _"http://purl.org/dc/elements/1.1#",
  foaf _"http://xmlns.com/foaf/0.1#",
  wsml _"http://www.wsmo.org/wsml-syntax#",
  loc _"http://www.wsmo.org/ontologies/location#",
  oo _"http://example.org/ooMediator#"

ontology _"http://www.example.org/ontologies/example"
  namespace _"http://www.example.org/ontologies/example#",
  dc _"http://purl.org/dc/elements/1.1#",
  foaf _"http://xmlns.com/foaf/0.1#",
  wsml _"http://www.wsmo.org/wsml-syntax#",
  loc _"http://www.wsmo.org/ontologies/location#",
  oo _"http://example.org/ooMediator#"

concept Human
  nonFunctionalProperties
  
*/

*/
```

104
Listing 3-3 shows an example of the WSML syntax and Web Service elements.
Listing 3-3 Examples of WSML Web Service Elements

```xml
webService _"http://example.org/Germany/BirthRegistration"
  nf
    dc#title hasValue "Birth registration service for Germany"
    dc#type hasValue _"http://www.wsmo.org/TR/d2/v1.2/#services"
    wsml#version hasValue "$Revision: 1.1 $"
  endnf
  importsOntology { _"http://www.example.org/ontologies/example",
    _"http://www.wsmo.org/ontologies/location" }
  capability _"http://example.org/Germany/BirthRegistration#cap1"
    sharedVariables ?child
      precondition
        nonFunctionalProperties
          dc#description hasValue "The input has to be boy or a girl
          with birthdate in the past and be born in Germany."
        endNonFunctionalProperties
        definedBy
          ?child memberOf Child
          and ?child[hasBirthdate hasValue ?birthdate]
          and wsml#dateLessThan(?birthdate,wsml#currentDate())
          and ?child[hasBirthplace hasValue ?location]
          and ?location[locatedIn hasValue oo#de]
          or (?child[hasParent hasValue ?parent]
          and ?parent[hasCitizenship hasValue oo#de])
      assumption
        nonFunctionalProperties
          dc#description hasValue "The child is not dead"
        endNonFunctionalProperties
        definedBy
          ?child memberOf Child
          and naf ?child[hasObit hasValue ?x].
      effect
        nonFunctionalProperties
          dc#description hasValue "After the registration the child
          is a German citizen"
        endNonFunctionalProperties
        definedBy
          ?child memberOf Child
          and ?child[hasCitizenship hasValue oo#de].
  interface _"http://example.org/Germany/BirthRegistration#face1"
  choreography _"http://example.org/tobedone"
  orchestration _"http://example.org/tobedone"
```

### 3.4.3 OCML versus WSML

OCML and WSML have in common a frame-based style as well as constructs for supporting rules and Semantic Web Services.

OCML provides an operational environment, which is very efficient for building executable model based applications. More precisely, OCML combines logical specification constructs (e.g. `iff-def` relation option) with operational constructs (e.g.
functions) within knowledge models (ontologies), which alleviates the problem of using semantics for decision (e.g. selection of services) while allowing for the execution of component models (e.g. Goal achievement).

WSML is dependent on an external reasoner (e.g. IRIS) for performing instance validation and queries; and on an external execution environment (e.g. WSMX) for performing SWS related activities. By the time of this writing, this infrastructure for WSML was in development and still immature. Thus, it was not easy to build and test applications using the SWS concepts. Nevertheless, we used WSML (via IRIS) to model mapping rules via axioms (using function symbols) for the purpose of testing data mediation. In addition, we used the XML Schema format of WSML in order to experiment with an industrial-strength model transformation engine (see Chapter 5).

3.5 Discussion and Conclusions

The two contributions of this chapter are a conceptual framework for Semantic Web Services and IRS-III, which implements this conceptual framework. These contributions fulfil a lack in the area of a consolidate view and infrastructure for Semantic Web Services. The benefits of the presented SWS conceptual framework are twofold: first, this framework comprehensively defines the main aspects of applications using SWS technologies; second, this framework informs the implementation of a SWS infrastructure, which we followed in our implementation of IRS-III.
IRS-III follows on the idea of applying a knowledge modelling technique, which originated with the UPML framework and evolved into IRS-II, to develop business applications on the Web. With IRS-II we started changing the original IRS towards a more open and distributed framework by incorporating Web Services; and with IRS-III we have added more features required by business processes that need to interoperate. These features have mainly been acquired by adopting the WSMO conceptual model. Thus, in this chapter we have presented the IRS-III implementation focusing on the knowledge modelling aspects of three new activities: mediation, choreography and orchestration.

As mentioned in the beginning of this chapter, our contribution to the IRS-III framework included the definition and implementation of the mediation framework; the definition of the orchestration primitives (developed by another programmer); the development of the IRS-III browser and java API; and the development of the java Publisher and the WSDL Publisher. The complete specification and implementation of the Mediation Framework is given in the next chapter along with the description of a SWS development approach.
Chapter 4  A SWS Mediation Framework

In this chapter we define the task of mediation in the context of Semantic Web Services, and as our first contribution we describe our specification and proposed approach to mediation. Following on this specification, as our second contribution, we describe the implementation of a Mediation Framework as part of the IRS-III as introduced in Chapter 3, which defines components and mechanisms for handling mediation as part of different activities, such as discovery and composition. We also analyse the problem of supporting mediation under the application development perspective and as our last contribution in this chapter propose a technique for developing applications from Semantic Web Services.

4.1 The Problem of Mediation

Our analysis of existing Semantic Web Services approaches in Chapter 2 (Section 2.5) has shown that the SWS research area is still evolving along with the supporting technologies and that mediation aspects are either underspecified or overlooked. A summary of the research problem concerning mediation approaches that will undertake was described in Chapter 2, Section 2.11.

Historically, data and process heterogeneity has been a known problem in the arena of software system development, affecting the interoperability and integration of applications within and between enterprises; and more recently in the context of the Semantic Web. In software systems, mediation is needed to enable data exchange and process interaction in B2B scenarios, such as in information management systems.
(Wiederhold, 1992; Sheth, 1998) and business process management systems (Leymann and Roller, 2005; Aaslt and Hee, 2002). In the Semantic Web, mediation is needed to solve heterogeneities among ontologies, as in the area of ontology matching (Noy, 2000; Maedche et al., 2002; Kotis et al., 2004; Euzenat, 2004).

We can view Semantic Web Services as a new step beyond current data and process integration approaches, because they combine the autonomy and reusability of Web Services and the semantic interoperability provided by ontologies in the Semantic Web. However, as a combination of Semantic Web technologies and Web Service technologies, Semantic Web Services also inherit the heterogeneity problems that arise when two or more ontologies or two or more services have to be combined during business-to-business or enterprise application integration.

From a business viewpoint, integrating services provided by partner organisations into software applications has a high cost, because it means manually adapting the company’s system interface to the data and message formats of each new service provider. For example, in a business scenario where an online book seller (e.g. Amazon.com) has to interact with different book suppliers, the involved partners must not only adhere to the communication standards (e.g. SOAP) but also comply with the data schema (e.g. customer_id, book_ref, etc) as well as the operation invocation order (e.g. login, search, buy, etc) of their provided services. Although the advent of Web Services made the integration of services a far easier task, it still requires a request to be mapped to the interface requirements of available services.
Chapter 4 A SWS Mediation Framework

At the technical level, a software developer would typically follow several steps in order to implement mediation for business applications: a) checking whether two Web services or processes are compatible, which means that the messages exchanged should match; b) deciding how to handle incompatibilities, which typically involves building a mediator between the processes (otherwise changing the processes); c) deciding how to implement the mediation, which involves a combination of techniques to generate data mappings and to combine business activities from the participant processes; and d) implementing the application according to the previous steps using a compliant execution environment. In this thesis we will overlook step (a) since mediation is in fact applied after this step in case of existing incompatibilities, and focus on the remaining steps, assuming that mediation will be needed and supported from the viewpoint of semantic interoperability in the context of Semantic Web Services.

According to the above, we can view our research problems at the business level and at the technical level as follows:

- Business applications on the Web need to interoperate across domains and organisations (B2B scenario). Organisations require flexibility to work with new business partners (clients or service providers) and agility when market requirements change. Heterogeneity problems arise when data and services have to be combined during application or process integration.
• Semantic Web Services enables the semantic interoperability of services over the Web. Mediation is needed to enable the integration of data, and the automation of discovery, composition and invocation of services provided by different providers. Mediation is used to solve data and process heterogeneities. There are mismatches of data formats and process behaviour.

From the discussion above we define mediation in the context of Semantic Web Services as the task of resolving data and process mismatches during the interaction between a service provider and a service requester using ontological service descriptions. We refer to semantic data mediation when mismatches between ontology elements have to be resolved so that services can interoperate. We refer to process mediation when mismatches on the interaction behaviour of processes have to be resolved, thus affecting the service composition or the modelling of a process workflow.

4.2 A Knowledge Modelling based Approach to Mediation

We propose a knowledge modelling approach to mediation in the context of Semantic Web Services, which builds on requirements both for integrating services from different providers and for solving mismatches between different knowledge domains. We focus our approach on system to system interactions instead of human to system interfaces, thus facilitating the work of software developers in building business-to-business applications.

Our approach to mediation is motivated by the fact that we can harness ontologies and knowledge modelling techniques for the modelling and execution of business software
applications. In particular, ontological descriptions can be used to explicitly represent mappings and relations between Semantic Web Service elements, which can be used at runtime. There are various factors motivating the modelling of mediators for Semantic Web Services. First, the mediation provider (who provides mediation functions and mappings) may be independent of the service provider. Second, the mediator model makes a Web Service adaptable to different goals and processes, with no changes to the Web Service. Finally, we can use a reasoner to search for mediator descriptions together with related Goals and Web Services available in libraries/registries during design time and runtime.

Within the scope of our work, we are less interested in algorithms which can automatically detect whether two representations match (syntactically or semantically) and more interested in finding ways to explicitly represent mediation aspects which facilitate the integration of heterogeneous data and processes. We exploit the use of mapping rules or axioms between ontology elements and the use of mediation services between service descriptions as mediation mechanisms. We then use these mediation mechanisms as part of the bridges connecting between the requester representation of a service (a Goal) and the provider representation (a Web Service). Similarly, in business process modelling (more on this in Chapter 5), we can model mediation tasks that use those mediation mechanisms in order to map between process interaction activities (e.g. Receive, Send, Invoke).
More precisely, the mediation approach we propose is part of the IRS-III framework (Chapter 3, Section 3.3.1). IRS-III is primarily based on the knowledge modelling approach introduced in (Motta, 1999; Omelayenko et al., 2003). We also take into consideration the separation of concerns between Web Services and Mediators (Wiederhold, 1992; Fensel et al., 2006) by adopting the WSMO conceptual model. We aim at the creation of ontology models that can be used within discovery and composition of Semantic Web Services. In particular, as we will describe later in this chapter, our approach consists of providing declarative models of specialized mediators, which can refer to mapping rules or mediation services, for solving different types of mismatches. The Mediator models together with the Goals and Web Services models are the top-level elements of the IRS-III framework. A design-time tool may be used by developers to generate and store semantic descriptions according to these models.

4.3 A Specification for Mediators within SWS

In this section we abstract from the IRS-III implementation and provide a specification for mediators within Semantic Web Services using the top-level elements of Goals, Web Services, Mediators and Ontologies (as presented in Chapter 3, Section 3.3.3). We present a generic SWS application scenario through which we discuss a number of cases for mediation in order to define the operational semantics of a number of mediators.

We use the following definitions from our discussion in the previous sections for the purpose of terminology alignment.
Definition 1. Semantic Web Services are Web Services enriched with ontological descriptions of their capabilities. As a result, we can use Semantic Web Services to build business applications or compose them into business processes.

Definition 2. Mediation in the context of SWS is the task of resolving data and process mismatches during the interaction between a service provider and a service requester, using ontological service descriptions. We assume that heterogeneity problems arise when one or more ontologies or one or more services have to be combined during application or process integration.

Definition 2a. Following on definition 2, we refer to Data Mediation when mismatches between ontology elements have to be resolved so that services can interoperate.

Definition 2b. Following on definition 2, we refer to Process Mediation when mismatches on the interaction behaviour between processes have to be resolved, affecting the modelling of the mediation process’ workflow.

We envisage a data-oriented business scenario, as illustrated in Figure 4-1, where a Web application makes invocation requests according to Goal descriptions that can be achieved by one of a set of applicable Web services. A Web Service WS is applicable to a Goal G if there is a Mediator M whose source is G and target is WS. Via discovery, an applicable Web Service may be selected for invocation if the constraints are satisfied for the input given. In this scenario, the application orchestrates the Goals to be invoked according to user inputs. This picture does not illustrate the cases of composition of Goals.
by the application or by Web Services for the purpose of clarity. Yet, the composition of Goals can be modelled via the Web Service’s orchestration description in order to accomplish a specific functionality. The dotted arrows indicate to what objects the semantic descriptions refer. That is, Web Service descriptions are used to annotate WS APIs and Goal descriptions are used to annotate client requests. WS APIs represent here syntactic interface descriptions for deployed Web Services using some standard such as WSDL or REST. The thin arrows indicate which domain ontologies the semantic descriptions use. The large arrows indicate a connection through mediators.

![Figure 4-1 Illustration of cases for Mediation in the context of SWS](image)

From the scenario above, we illustrate a number of cases for mediation according to mediator descriptions $M$ between ontologies $O$, Web Service descriptions $WS$ and Goal descriptions $G$. Consider $WS\ API_1$ and $WS\ API_2$ as examples of APIs to which WS descriptions are grounded (syntactically); and *Citizen* and *Customer* as examples of ontology concepts. We consider the following cases of mediation:
i. Goal $G_1$ and Web Service $WS_1$ represent request and provider capabilities by different concepts, say *Citizen* and *Customer* in ontologies $O_1$ and $O_2$ respectively. In this case, the requester and the provider have different semantic views over the same syntactic terms. $WS_1$ uses Mediator $M_1$ to align ontologies $O_1$ and $O_2$; and Mediator $M_2$ is used to express that $WS_1$ can solve $G_1$.

ii. Goal $G_1$ and Web Service $WS_2$ represent request and provider capabilities by different concepts, say *Citizen* and *Customer* in ontologies $O_1$ and $O_2$ respectively. Thus, $WS_2$ reuses Mediator $M_1$ to align ontologies $O_1$ and $O_2$. Mediator $M_3$ is used to express that $WS_2$ can solve $G_1$. Note, however that $WS_2$ is grounded to a different API - $WS API_2$, (requiring a different lifting/lowering function from $WS_1$). In this case, two providers ($WS_1$ and $WS_2$) have the same semantic view over different syntactic terms.

iii. Goal $G_1$ and Web Service $WS_3$ represent request and provider capabilities by the same concepts, say *Citizen* in ontology $O_1$. There is no need for capability alignment. Mediator $M_4$ is used to express that $WS_3$ can solve $G_1$. Additionally, we may use $M_3$ to carry out a transformation between input values (for example in case $WS_3$ use input values in different units from $G_1$). Note, however, that although $WS_3$ is grounded to the same API - $WS API_2$ – as $WS_2$, they have different semantic views over the same syntactic term (requiring different lifting/lowering functions).
iv. Different Goals (not illustrated) may be combined (e.g. through a workflow construct) to solve the functionality of a composite Web Service. We can use a Mediator to transfer and maybe transform data from one Goal to another. Moreover, we can use another Mediator to align different request capabilities.

v. Similar to the case above, different Web Services (not illustrated) may be combined (e.g. through a workflow construct) to solve the functionality of a composite Web Service. We can use a Mediator to transfer and maybe transform data from one Web Service to another. Moreover, we can use another Mediator to align different provider capabilities.

According to the cases above we can specify the operational semantics of five types of mediators in the form $sM_t$, where $S$ is the source of the mediator and $T$ is the target of the mediator in respect to the data flow, as follows:

a. $o_pM_{o_q}$ maps elements of $O_p$ to elements of $O_q$. This mediator is used to align instances of classes and relations between $O_p$ and $O_q$. If $p=q$ then the elements are in the same ontology. This mediator can be associated with a Goal or Web Service. This mediator is referred to as **OO-mediator**.

b. $g_pM_{w_s q}$ maps input elements of $G_p$ denoted by $G^{IN}_p$ to input elements of $W_S q$, denoted by $WS^{IN}_q$. This mediator is used to send and map data to Web Services when a Goal request is formed. This mediator is referred to as **WG-mediator**.
c. \( \text{WSqM}_{Gp} \) maps the output element of \( \text{WSq} \) denoted by \( WS^\text{OUT}_q \) to the output element of \( Gp \), denoted by \( G^\text{OUT}_p \). This mediator is used to send and map data to the Goal when a response is produced by a Web Service. This mediator is also referred to as \text{WG-mediator}.

d. \( \text{GpM}_{Gq} \) maps the input or output elements of \( Gp \), denoted by \( G^\text{IN}_p \) and \( G^\text{OUT}_p \) respectively to the input elements of \( Gq \), denoted by \( G^\text{IN}_q \). This mediator can be associated with Goals within compositions. This mediator is referred to as \text{GG-mediator}.

e. \( \text{WSpM}_{WSq} \) maps the input or output elements of \( \text{WSp} \), denoted by \( WS^\text{IN}_p \) and \( WS^\text{OUT}_p \) respectively to the input elements of \( \text{WSq} \), denoted by \( WS^\text{IN}_q \). This mediator can be associated with Web Services within compositions. This mediator is referred to as \text{WW-mediator}.

These mediators have been derived from the WSMO mediators as described in Chapter 2, Section 2.5.3 (OO-mediator, WG-mediator, WW-mediator, GG-mediator), however, the WSMO definitions are very generic while the mediators here have been given a more precise operational semantics. Note that for compatibility with WSMO, the two types of WG-mediator are distinguished according to the source and target components (Web Service or Goal).
In our implementation we make further distinctions from the WSMO mediators. First, we do not define chains of mediators as this would require a control mechanism; instead we allow for the definition of one or more mediators for the same target and source, carrying out different mappings. Second, we restrict the source and target of OO-mediators to be only ontologies, but they can be used by (or associated to) either a Web Service or a Goal. Moreover, we associate mapping rules to the \( \text{OpM}o_q \) mediator and a Mediation Service to the remaining types of mediators. A Mediation Service can be implemented just as a Goal in order to perform transformations between inputs and outputs. We propose an implementation for ontology mappings and Mediation Services in 4.5.1.

As indicated above, these mediators can provide mediated data-flow between a Goal and a Web service, and between Goals or between Web Services. Hence, we assume that in our framework inputs and outputs associated with a Web Service \( WS_q \) or a Goal \( G_p \), are explicitly modelled and denoted by \( WS_q^{IN} \), \( WS_q^{OUT} \), \( G_p^{IN} \) and \( G_p^{OUT} \).

### 4.3.1 Runtime Mediated Discovery

Discovery within Semantic Web Services can be viewed as the task of finding one or more services that satisfy a request (goal). In a mediation system (or brokering platform) based on goal achievement, discovery can be performed at runtime by matching the semantic description of a request against the semantic descriptions of applicable services, whereas we assume that one will be selected according to a selection algorithm (which might be as trivial as choosing the first match).
In a typical solution, first, the input and output types must match, then the preconditions must hold true for the input values given. Moreover, other descriptions such as non-functional properties should also be matched. However, assuming the requirements of business applications, we claim that service providers participating in an integrated system are not always compatible. Hence precise mappings must be provided in order to communicate with heterogeneous partners. Thus, the problem of discovery as finding relations between input types (e.g. Bouquet et al., 2004; Stollberg et al., 2006) is not applicable here, because we assume that mappings can be provided even when inputs do not match. That means an interaction may be desired despite the presence of heterogeneities.

To support runtime discovery, we must create a more controlled search space where for a given Goal $G_1$ we define mediators $G_1 M_{WSq}$ such that $WSq$, for $q=1..n$ are the set of applicable Web Services. We can then support runtime mediated discovery as follows: a) search for all mediators for which $G_1$ is the source, finding the set of applicable Web Services (targets); b) for every mediator, apply the corresponding mappings or transformations; and c) check whether the preconditions and non-functional properties of applicable Web Services hold true for the mapped values; d) select the first (or best) match from above. We have implemented this discovery algorithm as part of the mediation framework of IRS-III (Section 4.5).
4.3.2 Runtime Mediated Composition

Composition within Semantic Web Services can be viewed as the task of combining one or more atomic services so that together they achieve a desired functionality. Here we focus on business scenarios in which the composed services are executed as a workflow (orchestration model) and propose the use of mediators to map between atomic activities within and between partner processes. Our approach differs from approaches that try to derive (or synthesize) the workflow automatically (e.g. Traverso and Pistore, 2004; Wu, Z. et al., 2007) as we essentially aim at the modelling of mediators as part of process workflows.

To support process mediation, we make use of mediators of the form gMg and oMo. An orchestration model using Goal-based activities, may use those mediators such that for two goals Gp and Gq, GpMq transforms the output of the source Goal denoted by \( G_p^{OUT} \) to the inputs of the target Goal denoted by \( G_q^{IN} \). We have implemented this composition approach as part of the orchestration mechanism of IRS-III in Section 4.5.

4.4 Requirements for a SWS Mediation Framework

We present in the following the requirements for mediation within a Semantic Web Service infrastructure, derived from a number of research problems summarized in Chapter 2 (Section 2.6.4).
Mediation between the client application and the provided services: the SWS broker

In Chapter 3 we described a number of activities which are performed during the life cycle of a SWS based application, such as publishing, selection and composition. Traditionally, for syntactically described Web Services, these activities can be supported programmatically, but with the intervention of the developer who has to interpret and combine non-matching services. With SWS, by providing a semantic description of a Web Service we enable a software broker to use the knowledge available for managing those activities. At the highest level, we can define mediation as the brokering between a client software application and service providers. This definition follows on the conceptual framework from (Wiederhold, 1992), whereby activities that are distinguished from those carried out by the client or the server systems may be separated in a middleware component, the mediator. We can see the SWS broker as assuming this mediator role, by including specialized mediation components or handlers that provide mediation supporting the activities of discovery, composition and invocation while solving mismatches at the semantic level.

Mediation across domain ontologies: the data mediator component

Semantic data mediation tackles the problem of alignment between ontologies associated with data resources. This problem alone is one of the main research topics on ontology matching and merging (e.g. Noy and Musen, 2000; Maedche et al., 2002; Kotis et al.,
2004; Euzenat, 2004) in the Semantic Web, which investigates solutions in terms of automatically or semi-automatically generating declarative mappings between different ontological elements.

Within a SWS infrastructure, semantic descriptions are used whenever a service is queried or invoked. A developer might want to make available the mappings and transformations between elements representing different aspects of the service to support dataflow of composed services. It might be also necessary to transform inputs during the selection and invocation process. As a result, data mediator models are necessary to describe mappings between ontologies to be used by other elements such as Goals and Web Services or when importing ontologies. In addition, the data mediator model can refer to a mediation service, designed to perform generic types of transformations on behalf of the service, for instance concatenations or splitting of input or output values. Thus, components of the run-time environment should have access to semantic descriptions from the running service or process in order to solve existing mismatches.

**Mediating the invocation and orchestration of SWS: the process mediator component**

One of the uses of Semantic Web Services in software applications is to represent interaction activities associated with business processes belonging to organisations. The composition and invocation of such SWS for fulfilling a specific process must deal with the interaction behaviour of the corresponding services and mismatches that may occur.
When two processes must interact, mismatches can occur for example between the format or the order in which the information is requested and the way in which information is provided. In this case, the mediator process will handle the interaction protocol, which can be expressed in terms of interaction activities through the orchestration or process workflow model, representing the composition of SWS.

**Mediating the selection of SWS: the goal mediator component**

Typically, a search for applicable services will be performed during design time, when providers must be chosen. In this case the semantic descriptions are useful for matching based on concepts (or types) and subclassing relationships. However, during runtime an automatic selection based on goal input values may need mediators in order to satisfy the constraints from the provider side. The mediator component will handle the selection of applicable services when a goal achievement is requested.

**4.5 Implementation of the Mediation Framework**

In this section we describe the implementation of the mediation framework as part of IRS-III, using OCML (see Chapter 3, Section 3.4.1). We describe the mediator model and specific mediator types, the specification of the two mediation mechanisms, which are mapping rules and mediation services; and the architecture components. This description extends the description of IRS-III from Chapter 3, following the same design principles.
4.5.1 The Mediator Model

We describe the Mediator model of IRS-III in Listing 4-1, which is an ontology for the types of mediators specified previously (Section 4.3). The complete model can be found in Appendix A. The main characteristic of this model is that it is defined at the meta-level (like Web Services and Goals). This is enabled by meta-class definitions (e.g. meta-OO-mediator), which define subclasses of mediators as mediators. Hence, we can define individual mediators as classes whose values refer to other SWS elements according to the types (:\textit{type}) specified in the mediator meta-model. We will refer to these individual mediators as mediator descriptions or just Mediators. Also, these mediator descriptions are stored in libraries and instantiated at runtime.

The main class \textit{Mediator} is sub-classed into more specific types of mediators: \textit{OO-mediator, WG-mediator, WW-mediator, GG-mediator}. The mediators differ according to the type of source and target components they can handle and whether it uses a \textit{mediation service} or \textit{mapping rules}.

\textbf{Listing 4-1 IRS-III mediator model}

```lisp
(def-class meta-mediator () ?x
  :iff-def (or (= ?x mediator)
              (subclass-of ?x mediator)))

(def-class mediator (wsmo-entity))

(def-class meta-oo-mediator (meta-mediator) ?x
  :iff-def (or (= ?x oo-mediator)
              (subclass-of ?x oo-mediator)))

(def-class meta-ontology (wsmo-entity))

(def-class mapping-rules (list))

(def-class oo-mediator (mediator)
  ((has-mapping-rules :type mapping-rules :cardinality 1)
   (has-source-component :type meta-ontology)))
```
In the following we describe in more detail each mediator type, describe examples of mediator descriptions in OCML and describe how mapping rules and mediation services are constructed.

**OO-mediator**

An OO-mediator defines a source component, a target component and mapping rules. The source and target components are ontologies. The mapping rules are described below. An OO-mediator can be used by a Goal or Web Service (used-mediator attribute).

For instance, as shown in Figure 4-2, when OO-mediator OOi is used, the mapping rules are applied in order to map elements of Oi to elements of O2.
An example of an OO-mediator description in OCML is given in Listing 4-2. This example is taken from a use case in Chapter 6, Section 6.2. The OO-mediator in this example is named redirect-equipment-goal-OO-mediator. The ontology change-of-circumstances-prototype is used both as source (has-source-component attribute) and target (has-target-component attribute) since the mapping rules refer to elements of Goal descriptions within the same ontology (see appendix G). The mapping between elements is done via the mapping rule (has-mapping-rules attribute) named redirect-address-mapping. In the example, we use this rule to map input (attribute) values from Goal change-address-goal into the input (attribute) value new_address of Goal redirect-equipment-goal. The input values are captured via variables (e.g. ?pn) and concatenated into one value (?na) using the OCML function string-append.

During runtime, the source and target ontologies are imported into a temporary ontology (together with the mapping rules), which is discarded after the Web Service invocation. We explain next how Mapping Rules are constructed in OCML.
Listing 4-2 Example of an \textit{OO-mediator} description in OCML

\begin{verbatim}
(DEF-CLASS REDIRECT-EQUIPMENT-GOAL-OO-MEDIATOR (OO-MEDIATOR) ?MEDIATOR
  ((HAS-SOURCE-COMPONENT :VALUE CHANGE-OF-CIRCUMSTANCES-PROTOTYPE)
   (HAS-TARGET-COMPONENT :VALUE CHANGE-OF-CIRCUMSTANCES-PROTOTYPE)
   (HAS-MAPPING-RULES :VALUE
     (((redirect-address-mapping
       (NEW_ADDRESS ?REDIRECT-EQUIPMENT-GOAL ?na)
       IF
       (CHANGE-ADDRESS-GOAL ?CHANGE-ADDRESS-GOAL)
       (PREMISE_NUMBER ?CHANGE-ADDRESS-GOAL ?pn)
       (STREET ?CHANGE-ADDRESS-GOAL ?s)
       (CITIZEN_TOWN ?CHANGE-ADDRESS-GOAL ?t)
       (POST_CODE ?CHANGE-ADDRESS-GOAL ?pc)
       (= ?na (string-append ?pn " 11 1 1 11 11 11" ?s " 11 1 1 11 11 11" ?t " 11 1 1 11 11 11" ?pc))
     )))))))
\end{verbatim}

\textbf{Mapping Rules}

The implementation of Mapping Rules is based on Backward Rules from OCML (see also Chapter 3, Section 3.4.1). Backward rules are a mechanism for refining the space of values associated with relation entities in knowledge models (Motta, 1999). Note that in OCML classes are defined as unary relations, and attributes (slots) are defined as binary relations. A backward rule provides an expression which is sufficient to verify that a tuple (instance) holds for a relation, as shown in the example in Listing 4-3. In this example, the backward rule \textit{map-name-value-rule} is used to associate the \textit{has-name} attribute value of class \textit{Person} to the \textit{has-name} attribute value of class \textit{Citizen}, assuming that the classes have been defined.

Listing 4-3 Example of an OCML backward rule

\begin{verbatim}
(deffuns map-name-value-rule
  ((has-name ?citizen ?name)
   if
   ((Citizen ?citizen)
    (Person ?person)
    (has-name ?person ?name))))
\end{verbatim}

129
More precisely, a backward rule consists of a number of backward clauses, defined according to the following syntax:

\[
\text{backward-rule ::= (def-rule rule-name backward-clause+)}
\]

\[
\text{backward-clause ::= (relation-expression if \{logical-expression+\})}
\]

where, \text{def-rule} and \text{if} are keywords in OCML. The parentheses represent lists in OCML. The plus sign is used to indicate that the term can appear one or more times. The curly brackets are used to indicate that the enclosed term is optional. \text{Relation-expression} and \text{logical-expression} are (self-explanatory) terms allowing the use of variables, as exemplified before. When carrying out a proof by means of a backward rule, the OCML interpreter will try to prove whether some tuple is an instance of the relevant relation by firing the clauses in the order in which these are listed in the rule definition. As in Prolog, depth-first search with chronological backtracking is used to control the proof process.

From the definition above, we define a mapping rule according to the following syntax:

\[
\text{mapping-rule ::= (mapping-name backward-clause+)}
\]

Therefore, we use backward rules as a simple way to construct mapping rules. As a result, we can define the \text{has-mapping-rules} attribute in the OO-mediator description as a list of one or more mapping rules. During runtime, the mapping rules from the \text{OO-mediator} are converted into backward rules within a temporary ontology (\text{mapping-name} becomes the \text{rule-name} of a backward rule). A mapping-rule is used when a query
involving the relation-expression is performed. In the example shown in Listing 4-2, when IRS-III queries the attribute new_address, this is resolved via the mapping rule.

**WG-mediator**

A WG-mediator defines a source component, a target component and a Mediation Service. A WG-Mediator is used for connecting a Web Service (target) to a Goal (source) it can achieve, resolving mismatches. The Mediation Service is defined as a Goal, which is used to transform data between the connected Web Service and Goal. Figure 4-3 shows a graphical illustration of the mediation between Goal G1 (the source) and Web Service WS1 (the target) via WG-mediator WGi. The Mediation Service associated with WGi is Goal G2, which means it gets achieved when WGi is used. G2 is used to transform the input value I1 of G1 into the input value I2 of WS1.

![Figure 4-3 A WG-mediator diagram](image)

The Mediation Service G2 must also follow some rules, in order to enable data flow: either input I1 must have the same name in G1 and G2 or an OO-mediator must be provided to map between inputs. The same applies to I2 between G2 and WS1. The reasoner behind the scenes will match the input and output names.
Note that for the purpose of reusability, if the target element is not present in a Mediator description, IRS-III assumes that there may be a Web Service using (via the used-mediator attribute) that mediator, so that they are considered the targets. In addition, when the input and output elements of a Web Service are not present, they are automatically inherited from the Goal to which the Web Service is linked (via a WG-Mediator).

Mediation Services are used (instead of mapping rules) when values need to be transformed rather than interpreted in a different way. For example, in Listing 4-4 we show the instance of a WG-Mediator (universal-time-buy-train-ticket-mediator), which uses a Mediation Service to transform the value of time given in a list format (has-list-and-date-time) to a value in string format (has-universal-date-and-time). The Mediator’s source Goal (buy-train-ticket-goal) and target Web service (german-buy-train-ticket-service) are given in Listing 4-5. Note that the target component is not present in the mediator instance, but instead the one declared in the Web Service’s capability via the used-mediator attribute (slot) is used. This example is specified within a scenario in which a user can buy train tickets in Europe; and has been provided as part of a tutorial on SWS using IRS-III, available at http://kmi.open.ac.uk/projects/dip/resources/AAAI06/AAAI06.zip.

**Listing 4-4** Example of a *WG-mediator* and a Mediation Service

```lisp
(DEF-CLASS UNIVERSAL-TIME-BUY-TRAIN-TICKET-MEDIATOR (WG-MEDIATOR)
  (HAS-SOURCE-COMPONENT :VALUE BUY-TRAIN-TICKET-GOAL)
  (HAS-MEDIATION-SERVICE :VALUE UNIVERSAL-TIME-BUY-TRAIN-TICKET-MEDIATION-GOAL )))
```
Listing 4-5 Example of a Source Goal and a Target Web Service for the WG-Mediator

(DEF-CLASS UNIVERSAL-TIME-BUY-TRAIN-TICKET-MEDIATION-GOAL (GOAL) ?GOAL
  (HAS-INPUT-ROLE :VALUE HAS-LIST-DATE-AND-TIME)
  (HAS-INPUT-SOAP-BINDING :VALUE (HAS-LIST-DATE-AND-TIME "sexp"))
  (HAS-OUTPUT-ROLE :VALUE HAS-UNIVERSAL-DATE-AND-TIME)
  (HAS-OUTPUT-SOAP-BINDING :VALUE (HAS-UNIVERSAL-DATE-AND-TIME "string"))
  (HAS-LIST-DATE-AND-TIME :TYPE LIST)
  (HAS-UNIVERSAL-DATE-AND-TIME :TYPE STRING)))

(Listing 4-5 continues)

(DEF-CLASS BUY-TRAIN-TICKET-GOAL (GOAL) ?GOAL
  ((HAS-INPUT-ROLE :VALUE HAS-PERSON
    :VALUE HAS-DEPARTURE-STATION
    :VALUE HAS-DESTINATION-STATION
    :VALUE HAS-LIST-DATE-AND-TIME)
   (HAS-INPUT-SOAP-BINDING :VALUE (HAS-PERSON "sexpr")
    :VALUE (HAS-DEPARTURE-STATION "sexpr")
    :VALUE (HAS-DESTINATION-STATION "sexpr")
    :VALUE (HAS-LIST-DATE-AND-TIME "sexp"))
   (HAS-OUTPUT-ROLE :VALUE HAS-BOOKING-ORDER)
   (HAS-OUTPUT-SOAP-BINDING :VALUE (HAS-BOOKING-ORDER "string")
    :VALUE (HAS-PERSON :TYPE PERSON)
    :VALUE (HAS-DEPARTURE-STATION :TYPE CITY)
    :VALUE (HAS-DESTINATION-STATION :TYPE CITY)
    :VALUE (HAS-LIST-DATE-AND-TIME :TYPE LIST)
    :VALUE (HAS-BOOKING-ORDER :TYPE STRING)
    :VALUE (HAS-NON-FUNCTIONAL-PROPERTIES
      :VALUE BUY-TRAIN-TICKET-GOAL-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS GERMAN-BUY-TRAIN-TICKET-SERVICE (WEB-SERVICE) ?WEB-SERVICE
  ((HAS-INPUT-ROLE :VALUE HAS-PERSON
    :VALUE HAS-DEPARTURE-STATION
    :VALUE HAS-DESTINATION-STATION
    :VALUE HAS-UNIVERSAL-DATE-AND-TIME)
   (HAS-OUTPUT-ROLE :VALUE HAS-BOOKING-ORDER)
   (HAS-DEPARTURE-STATION :TYPE CITY)
   (HAS-DESTINATION-STATION :TYPE CITY)
   (HAS-LIST-DATE-AND-TIME :TYPE LIST)
   (HAS-BOOKING-ORDER :TYPE STRING)
   (HAS-NON-FUNCTIONAL-PROPERTIES
    :VALUE GERMAN-BUY-TRAIN-TICKET-SERVICE-NON-FUNCTIONAL-PROPERTIES)
   (HAS-CAPABILITY :VALUE GERMAN-BUY-TRAIN-TICKET-CAPABILITY)
   (HAS-INTERFACE :VALUE GERMAN-BUY-TRAIN-TICKET-SERVICE-INTERFACE)))

(DEF-CLASS GERMAN-BUY-TRAIN-TICKET-CAPABILITY (CAPABILITY) ?CAPABILITY
  ((USED-MEDIATOR :VALUE UNIVERSAL-TIME-BUY-TRAIN-TICKET-MEDIATOR)
   (HAS-ASSUMPTION :VALUE
    (HAS-VALUE :VALUE "string"))
   (HAS-NON-FUNCTIONAL-PROPERTIES
    :VALUE GERMAN-BUY-TRAIN-TICKET-SERVICE-NON-FUNCTIONAL-PROPERTIES)))
GG-mediator

Similar to a WG-mediator, a GG-mediator defines a source component, a target component and a Mediation Service. In this case, both the source and target components are Goals. The Mediation Service is defined in the same way as for the WG-mediator (see previous section). A GG-Mediator is used to provide data-flow between two Goals within an orchestration, resolving mismatches.

The graphical diagram in Figure 4-4 illustrates the mediation between Goal $G_1$ (the source) and Goal $G_2$ (the target) via GG-mediator $GG_1$. The Mediation Service associated with $GG_1$ is also a Goal ($G_3$), which means it gets achieved when $GG_1$ is used. $G_3$ is used to transform the output value $I_1$ of $G_1$ into the input value $I_2$ of $G_2$. $I_1$ could also be an input of $G_1$, since Goal inputs receive values.

![Figure 4-4 A sample GG-mediator](image-url)
Listing 4-6 shows an example of GG-Mediator description taken from the use case in Chapter 5, which is similar to the WG-Mediator description given before. The redirect-equipment-goal-gg-mediator, via the redirect-equipment-goal-gg-mediator-mediation-service, transforms the input values of inputs premise_number, premise_name, street, citizen_town and post_code from change-address-goal into one concatenated (string) value for input new-address in redirect-equipment-goal.

Listing 4-6 redirect-equipment-goal-gg–mediator and Mediation Service Description

```lisp
(DBF-CLASS REDIRECT-EQUIPMENT-GOAL-GG—MEDIATOR (GG-MEDIATOR)
  ((HAS-SOURCE-COMPONENT
    :VALUE CHANGE-ADDRESS-GOAL)
   (HAS-TARGET-COMPONENT
    :VALUE REDIRECT-EQUIPMENT-GOAL)
   (HAS-MEDIATION-SERVICE
    :VALUE REDIRECT-EQUIPMENT-GOAL-GG—MEDIATOR-MEDIATION-SERVICE)))

(DBF-CLASS REDIRECT-EQUIPMENT-GOAL-GG—MEDIATOR-MEDIATION-SERVICE (GOAL) ?GOAL
  ((HAS-INPUT-ROLE
    :VALUE POST_CODE
    :VALUE PREMISE_NUMBER
    :VALUE PREMISE_NAME
    :VALUE STREET
    :VALUE CITIZEN_TOWN
    (HAS-INPUT-SOAP-BINDING
     :VALUE (POST_CODE "string")
     :VALUE (PREMISE_NUMBER "string")
     :VALUE (PREMISE_NAME "string")
     :VALUE (STREET "string")
     :VALUE (CITIZEN_TOWN "string"))
   (HAS-OUTPUT-ROLE
    :VALUE NEW_ADDRESS)
   (HAS-OUTPUT-SOAP-BINDING
    :VALUE (NEW_ADDRESS "string")
    (POST_CODE :TYPE STRING)
    (PREMISE_NUMBER :TYPE STRING)
    (PREMISE_NAME :TYPE STRING)
    (STREET :TYPE STRING)
    (CITIZEN_TOWN :TYPE STRING)
    (NEW_ADDRESS :TYPE string)))
```

4.5.2 Data and Process Mediation

IRS-III handles data mediation by executing mapping rules defined in the OO-mediator or a mediation service defined in the WG-mediator or GG-mediator. IRS-III does not
implement the \textit{WW-mediator} since we implemented a goal-based invocation design principle (see Chapter 3, Section 3.3.2). We explain in the section on architecture below how data mediation takes place at runtime.

IRS-III handles process mediation by executing the orchestration defined in the Web service description as explained in the section on architecture below (see also Chapter 3, Section 3.3.4). Process mediation in IRS-III is implemented as proof-of-concept to handle process activities as Goal invocations. The orchestration is a simple workflow model which consists of control-flow constructs, which can for example connect one or more Goals in sequence. An illustration is given in Chapter 6, Section 6.2.2.

\subsection*{4.5.3 Architecture}

Figure 4-5 illustrates the main architecture components of the mediation framework of IRS-III, which consists of the \textit{data mediator}, \textit{goal mediator} and \textit{process mediator}. The main objective is to solve different types of mismatches by reasoning over the given Goal, Web Service and Mediator descriptions. The implementation code of these components is provided in appendices B, C and D.

The goal mediator component of IRS-III handles mismatches that occur during the process of discovery of Web Services for solving a Goal, hence it implements the runtime mediated-discovery specification proposed earlier (Section 4.3.1), by applying WG-mediators.
The Process mediator component of IRS-III handles mismatches that occur during the orchestration of a Web Service, hence it implements the runtime mediated-composition specification proposed earlier (Section 4.3.2), by applying GG-mediators. During the invocation, IRS-III either executes the choreography (interaction protocol) of a single Web Service or the orchestration of a composed Web Service. The IRS-III choreography mechanism (see Chapter 3, Section 3.3.4) does not use mediators.

The Data Mediator component is used by the Goal Mediator and by the Process Mediator for mapping or transforming data instances. It executes mapping rules defined in the OO-mediator or a mediation service defined in the WG-mediator or GG-mediator.

In the steps below we describe the overall sequence of mediation activities taking place during selection, composition and invocation of Semantic Web Services.

- The Goal Mediator component searches for WG-mediators whose source component matches the Goal requested from a client application to find the set of applicable Web Services (target components). WG-mediators can be found in the
IRS-III library or attached to the capability of Web Services. The component selects the first Web-Service from this set, which matches the requested capabilities (input types, preconditions, assumptions, non-functional properties etc) for the input values given.

- For single Web Services, the Process Mediator component establishes an interaction with a deployed web service (code) by executing its Web Service choreography through the Choreography Interpreter, which creates the communication messages corresponding to the choreography communication primitives. The Process Mediator performs the lifting and lowering of data provided by the Web Service grounding and communicates with the Invoker component for operation calls.

- For composite Web Services, the Process Mediator component executes the orchestration using the Orchestration Interpreter. It keeps the state of the orchestration (control and data flow) between invocations of sub-Goals. The Process Mediator component searches and applies GG-mediators attached to Goals in the orchestration. The common types of mismatches that can occur are: output of a source goal does not match the input in the target Goal; the input or output of a source Goal has to be split or concatenated into the inputs of the target sub-goal.
4.5.4 Conclusions about the IRS-III Implementation

The implementation of mediation in IRS-III extends and integrates with the previous models and execution components of IRS-II. The main extension lies in defining the execution semantics of each type of mediator within the framework. In addition, it is necessary to define the (control-flow) primitives for orchestration taking into consideration the integration with mediators.

It is important to note that the previous implementation of SWS in IRS-II, assumed a very strong relation between Tasks and PSMs, that is, the PSM had to declare which Task it was going to solve. This implementation failed to make the view of the requester (Goal) independent of the view of the provider (Web Service). In IRS-III, although inheritance is allowed in case the inputs and outputs between a Goal and a Web Service are equivalent, the connections are implemented via mediators, so that a Web Service can be applicable to (solve) more than one Goal. The mismatches are solved by the mappings provided with the mediator.

The orchestration language developed in IRS-III is limited to a few constructs (see Chapter 3, Section 3.3.4) and implemented as proof-of-concept. Basically, the orchestration mechanism of IRS-III allows us to combine Goals in a structured manner using a few control-flow constructs (primitives), facilitating data-flow and solving mismatches through mediators. We understand that IRS-III is not the proper type of environment for the execution of business workflows. For example there is no...
mechanism in IRS-III for receiving messages from external services asynchronously. In the next chapter we explore another approach for process mediation with Semantic Web Services in the area of Semantic Business Process Modelling.

One important lesson we learned from using Mediator descriptions (see also the use case in Chapter 6, Section 6.2) at runtime is that by attaching a mediator to a Web Service or Goal description (via *used-mediator* attribute) we lose the reusability of that Web Service or Goal since that mediator cannot be used in every context (i.e. they are tied to a specific application). This makes us suggest to change the Web Service and Goal descriptions as described in the WSMO meta-model not to contain the *used-mediator* attribute. Instead, the mediation framework should search for the right mediator via its source and target attributes according to the SWS activity context (e.g. discovery, composition).

### 4.5.5 Mediator Model Reusability

Reusability is one of the main themes in Knowledge Modelling and more specifically in the IRS-III implementation. In this section we examine and outline two ways in which mediators can be reused: the Mediator Library and the Mapping Ontology.

**The Mediator Library**

The main purpose of the mediator library is to provide functionality to store new and access existing mediators. This functionality is necessary in order to search and reuse
existing mediators in new scenarios, minimizing the need for the custom creation of mediators.

In IRS-III we simply followed on the previous implementation of Goals and Web Services and extended the OCML library (presented in Chapter 3). Briefly, The Mediator Library enables users to search for mediators via a number of description slots including the mediator name, the mediator type, the source element, the target element, the mediation service as well as types of mappings (see Mapping Ontology below). The advantage of having the mediators available in a library is that not only the developer can search for existing ones (using the browser), but also the mediator framework can access the library during runtime.

The reusability rate of mediators is naturally not very high in the ontological sense, because mediators are designed for making source and target elements (Goals, Web Services and ontologies) reusable. On the other hand, it is very easy for the developer to create a new mediator, say based on a previous mediator with the same source or target. Also, the library makes it very easy for the developer to both inspect and replace mediation services and mapping rules.

**A Mapping Ontology**

We can also examine mediation from the viewpoint of ontology mapping within a business scenario and outline the use of a Mapping Ontology, which will help developers to search and reuse mappings.
In business scenarios using Web Services, it is common to define data types using XML Schema (see Chapter 2). These types represent the messages to be exchanged between the business applications and can be used as the syntactic grounding for one or more WSMO Web Service descriptions. Once the XML Schema is lifted to the ontological level, mediators containing declarative mapping rules can be created to represent the mappings to be executed at runtime.

![Mappings Ontology](http://dip.semanticweb.org)

Based on a small use case for mediation presented in the DIP project\(^{28}\), we gathered a number of examples from real world applications, which illustrate a bottom-up approach for mappings given two segments of XML Schemas. The two XML schemas used represented two different versions of a communication standard. It is common for business communication standards to be available in different versions. Also, most business communication standards offer a certain degree of flexibility, for example a

\(^{28}\) http://dip.semanticweb.org
delivery date element defined to be of type \textit{xsd:string} can allow the usage of ISO date formats as well as the usage of UK date formats.

We gathered the mappings requirements from the example above and then created a classification for mappings from which we can derive a Mapping Ontology (Figure 4-6). The Mapping Ontology would allow us to index the Mediator Library or create specific mapping primitives, according to the following mapping types:

- **Direct Mapping** – Uses a rule to map one (source) concept or relation (attribute) to another (target).

- **Conditional Mapping** – Same as a Direct mapping, but is only executed if the condition given is true.

- **Concatenation** – uses a mediation service (function) which concatenates the values of one or more concepts and generates the value of a target concept.

- **Instance Concatenation** – Same as concatenation, but works over all the instances of a source concept.

- **Conditional Concatenation** - Same as a Concatenation, but it is only executed if the condition given is true.

- **Splitting** – Inverse of Concatenation. Uses a mediation service (function) which splits the values of one concept into the values of one or more target concepts.
- Conditional Splitting - Same as a Splitting, but it is only executed if the condition given is true.

### 4.6 A SWS Application Development Approach

In this section we analyse the problem of supporting mediation through the software application development perspective and as a target result develop an approach for building applications from Semantic Web Services. In particular, we propose a generic web application architecture, which defines system development activities in a number of layers from data source access, to Web Service creation, to semantic annotation of Web Services using IRS-III, to user interface implementation.

We believe Semantic Web Services can be used in a variety of application systems and environments such as service-based Web applications, Semantic Search tools, domain specific Web applications, SOA-based development environments, Semantic Grid environments, Semantic Browsing, Business Process Management, Semantic Portals, to list a few. Semantic Web Services can be integrated into most of these application systems, however, it is important to have a generic view on how SWS fit in the application life cycle. In this chapter, we define a specific approach and show how developers could use the infrastructure provided in the previous sections. On the other hand, this approach is also useful as a basis for evaluating the infrastructure provided as it shows the steps to setup the infrastructure for a given application scenario.
We propose a generic web application architecture, which will guide developers with design-time tasks necessary to first build Semantic Web Services and then use them to build applications. In semantic terms, our guidelines are useful to ground business conceptualizations and knowledge to specific models (ontologies). More specifically, from an application requirements specification, the developer can: ground scenario description (context) vocabulary to domain ontologies; ground user requests to Goal descriptions; ground Web Service capabilities to Web Service descriptions; and ground mappings and transformations to Mediator descriptions. Having said that, this does not preclude the use of existing methods to extract vocabulary or build domain ontologies from textual sources (Gomez-Perez et al., 2003), or service APIs (Sabou et al., 2005) in order to get started. The same idea is valid for mediation, which might require that the developer use existing techniques to provide mappings between ontologies (Euzenat, 2004; Mocan, 2005).

Figure 4-7 A simplified SWS brokering scenario using IRS-III

We take the view of IRS-III as a semantic broker mediating between a service requester and one or more service providers. In this view, Web applications composed of Semantic Web Services can run by sending “achieve-goal” requests to IRS-III with the input values.
from the user. IRS-III will then invoke the appropriate deployed Web Services (Figure 4-7). This Semantic Web Service brokering scenario enables data and process integration across many business partners. The SWS provided can be shared or used to send common information to the diverse participating organisations.

In our methodology for developing applications using SWS with IRS-III we devise a customer team for creating goal descriptions according to user requests and a development team for creating web service descriptions for the available deployed Web Services. The application developer then creates mediator descriptions which connect domain ontologies, goals and web services and provide mediation services or mapping rules for solving mismatches between ontological elements.

A generic application architecture using IRS-III is depicted in Figure 4-8. Briefly, it enables the functionality provided by existing legacy systems from the involved business
partners to be exposed as Web Services, which are then semantically annotated and published using the SWS infrastructure. The architecture consists of four layers as explained next.

The legacy system layer consists of the existing data sources and information technology systems available from each of the parties involved in the integrated application. The service abstraction layer enables the functionality of the legacy systems to be available as Web Services, abstracting from the implementation details. Current Enterprise Application Integration (EAI) software generally enables the easy creation of the necessary Web Services. Note that for the integration of standard databases the necessary functionality of the Web Services can simply be implemented as query (SQL) functions. The SWS layer is based on the Web Services provided by the service abstraction layer. The activities in this layer are mainly supported by the IRS-III infrastructure as outlined in section 2. Given a goal request, IRS-III will: a) discover a candidate set of web services; b) select the most appropriate one; c) resolve any mismatches at the ontological level; and d) invoke the relevant set of Web Services satisfying any data, control flow and invocation requirements. To achieve this, IRS-III, utilizes the set of Semantic Web Service descriptions which are composed of goals, mediators, and web services, supported by relevant domain ontologies. Finally, the presentation layer consists of the user interface, which is built on top of the SWS layer as a Web application accessible using a standard Web browser. Goal requests are generated with the data provided by the
user through the user interface triggering the invocation of applicable SWS and as a result the execution of deployed Web Services in the service abstraction layer.

In general, during the requirements phase of application development, the stake holders involved in the application scenario should provide information to ontology builders in order to create or reuse domain ontologies related to the application context. SWS make this process very simple and efficient because the only knowledge which must be modelled is related to the exposed functionality implemented by the Web Services. Developers do not need to model entire data sources or create class instances corresponding to thousands of database records; we only model the information used by Web Services.

By taking a top-down approach for semantically annotating services, IRS-III facilitates querying and reasoning about the capability of the service before its execution since the semantic relations between the descriptions used (goal, web services, mediators and domain ontologies) are well defined in the WSMO metamodel. The reasoning needed during the invocation of one service is efficient because it is limited to the scope of the invocation.

4.7 Discussion and General Conclusions

This chapter is mainly devoted to the specification of mediation and to the development of software infrastructure supporting our approach. There are three main distinguishing aspects of our work worth recalling at this point in the thesis. First, the target audience or
user for this type of infrastructure is the software developer, which means that this infrastructure can only indirectly affect the experience of the final Web user (not in the scope of this thesis). Second, we constrain our approach to business scenarios, where there are a limited number of services provided and exchanged among an agreed number of participant business organisations. Third, although we provide infrastructure for developers, we aim at bridging the gap between the business view and the technical view of business applications.

We claim that our approach facilitates Semantic Web Service discovery, composition and execution in the presence of data and process heterogeneities. Moreover, this approach facilitates bridging the gap between business analysts’ and developers’ views, and makes it easier for developers to reuse and integrate mappings and services into new software applications.

Our solution is the only knowledge-level modelling (Motta, 1999) implementation to the mediation of SWS. Most approaches either do not model mediators (e.g. OWL-S, SAWSDL) or, in case of WSMO, do not use the mediator descriptions at runtime. Also, we do not attempt to provide an approach that generates mediators automatically. Instead, we use a model that can be executed in a knowledge-based environment such as IRS-III.

Although OWL-S does not model the mediator concept; yet, mediation plays a key role in the approach. The OWL-S approach considers that mediation is handled during discovery and decomposition by architectural components and that a mediation service is treated
just as another web service. This assumption makes mediation very implementation
dependent and not visible to the user.

The OWL-S process model (see Chapter 2, Section 2.5.2) has similar purpose to the
orchestration model of IRS-III in that it describes the workflow of a composite Web
Service, specified through a pre-defined set of control structures. The main difference is
that IRS-III uses Goals to accomplish process activities.

The WSMO approach (see Chapter 2, Section 2.5.3) specifies mediation, but the main
difference is that the Mediator conceptual model is not used by the mediation
components. The WSMX execution environment includes a data mediation component,
which can execute mapping rules generated at design time by a mapping tool; and a
process mediation component, which works on predefined types of mismatches between
two choreography instances at runtime. IRS-III instead, interprets the mediator models
for handling mismatches.

The SAWSDL\textsuperscript{29} standard (see details in Chapter 2, Section 2.6) prescribes a different
approach to SWS from IRS-III mainly because it is a bottom-up approach, which does
not define an ontological model for SWS. Rather, by using SAWSDL, one points to
ontologies and mapping functions provided externally. Ontology concepts are used to
annotate XML Schema type definitions of service's inputs and outputs as well as WSDL

\textsuperscript{29} \url{http://www.w3.org/2002/ws/SAWSDL}
operations. The mapping functions are used to translate between semantic data and XML. In IRS-III, instead, the mediators provide transformations at the semantic level, between semantic data. In addition, the Web Service *Grounding* description provides the translation between semantic data and syntactic data.

The fundamental difference between top-down approaches like IRS-III or OWL-S, and bottom-up approaches like SAWSDL is that top-down approaches have a fixed comprehensive service ontology for the annotation of any type of Web Services; while a bottom-up approach is tied to a syntactic standard (e.g. WSDL, REST, etc). Consequently, top-down approaches have to be concerned with the grounding to different Web Services while bottom-up approaches leave it up to the execution environment to reason over the referred ontologies.

The goal of Semantic Web Services is to provide semantic interoperability of services; however, current approaches overlook the problem of mediation, which affects the solutions for the automation of activities such as selection, composition and invocation. As more and more businesses are offered through the Web, which require flexibility and agility in finding and combining the services of partner organisations, mediation becomes crucial for solving heterogeneities. Software applications have to handle a number of mediating tasks in order to allow services from different providers to be selected and composed according to client goals or service requests.
As part of our research methodology, we have analysed the problem of mediation in the context of Semantic Web Services and provided a solution for mediation taking into consideration three perspectives: the SWS activities such as composition, discovery and invocation; ontology description in terms of the mediator meta-model and the business process ontology; and architecture such as our mediator framework, as proposed in the conceptual framework of Chapter 3.

In the work of Wiederhold (Wiederhold and Genesereth, 1997) mediation is about the clear definition of mediating tasks (or services). The main advantage of the introduction of mediators in system design is that code change is minimized and management facilitated when system updates are required in face of changes in the business goals. Although our approach is informed by the work of Wiederhold, our approach emphasizes modelling. The mediation model not only fits with the other SWS activities such as orchestration but it is also operational. The mediator descriptions are used within a semantic-based machinery (the OCML reasoner/proving system). This modelling approach to mediation also benefits system designers in reasoning over mediation aspects. This is in fact desired in business scenarios with incompatible partners and also for simulation and test purposes.

Our specification for mediated discovery and mediated composition takes into account the requirements of business organisations. In existing process mediation approaches aiming at automatic composition, a discovery mechanism must find the set of available services that can solve a number of predefined tasks (or goals). The problem with this is
that the services which do not match exactly the tasks (mainly inputs) are eliminated during the selection. In the real world, companies cannot eliminate clients or required service providers based on non-matching; rather they must adapt their interface. With mediators, we can not only provide mappings but also model the mediated composition without resorting to changes in the composing services or re-engineering of the process.

From the above we can see the benefits of our modelling-based approach to mediation. We can conclude that the emergence of innovative standards and technologies for application interoperability and integration requires a new role or set of roles for mediation in the world of open, distributed and service oriented information systems. Thus, mediation has to conform to the new context of automation of software systems integration in order to resolve heterogeneities between service requesters and service providers. Since the landmark paper on mediation from (Wiederhold, 1992), not only the technologies for software systems have changed, but also the roles of mediation in order to solve heterogeneities. Thus, Semantic Web Services as innovative technology for application interoperability require a new conceptualization of the mediation activity accordingly.
Chapter 5  Mediation and Business Process Modelling

In this chapter we contribute with the specification of the Business Process Modelling Ontology (BPMO) and describe its use for supporting mediation of workflow-based processes based on Semantic Web Services. In addition, we contribute with the implementation of a translator for transforming business-level process models using BPMO into executable process models.

5.1 Introduction

The work in this chapter complements and extends the work in the last chapter in what concerns process mediation. More precisely, in the last chapter we tackled process mediation using the orchestration mechanism present in IRS-III. We concluded then that IRS-III's orchestration model was restricted due to limitations in the infrastructure, in particular, IRS-III does not support unsynchronized message calls. This chapter is dedicated to the development of an ontology for fully fledged process workflows based on SWS, which can be used as the orchestration model for the purpose of mediation.

Composition or orchestration of Web Services is one of the predominant activities when building service-based business applications, as demonstrated by the number of existing composition languages available from industry (see Wohed et al., 2003). These languages enable the composition of Web Services into business processes and their automatic execution. Composition can thus be defined as the task of finding suitable Web Services for fulfilling business process workflow activities. We refer to
Process Mediation in this context when the composite (mediation) process is in fact controlling the invocation of Web Services from partner processes so that they can interact. Hence, the modelling of the mediation process is based on the combination of the control-flow, data-flow and atomic activities of the two or more processes that are being mediated. See also the description of related work in Chapter 2, Section 2.8.

More precisely, the subtle difference between process composition and process mediation is that in process mediation we combine two or more existing processes, while in process composition we just combine activities. Therefore, the workflow model of a mediation process can be generated from the workflow models of two (or more) existing processes. Note that what can make the two notions confusing is the fact that process activities can be atomic or composite. Usually, business-level processes may contain composite activities, which are themselves sub-processes. Execution-level processes, on the other hand, usually only contain atomic activities (e.g. invocation, request and reply). In addition, Web Services can be used either to represent (expose) processes or to represent atomic activities (assuming that a specific operation is specified).

Given the context above, by providing ontological descriptions of fully-fledged business processes models we can facilitate process mediation, as those descriptions enable the reasoning over the types and order of activities as well as the provision of semantic data types and mappings. It is worth noting that typically, it is the mediation process that is executed for resolving heterogeneities between of one or more partner processes in a business context.
Currently, in the area of Business Process Management (BPM), business analysts use process modelling notations such as BPMN\textsuperscript{30} and EPCs (Scheer et al., 2005) to define business process models as part of tool suites for Business Process Modelling (BPM). These notations are useful at the business level, but alone they provide no inference reasoning over the business process. For example, BPMN tools are rich in control-flow constructs, but the graphical elements contain only limited textual information with no formal semantics. EPC-based tools such as ARIS (Scheer et al., 2005) on the other hand, provide integration of different views (e.g. organisation, data and control) and levels (e.g. requirements and implementation); however mediating between these views and levels is a very complex task due to the variety of underlying representation formats. In addition, it is very difficult to use these notations to automatically query the business context or draw relations between existing processes or services.

More specifically, our work in this chapter has been developed as part of the SUPER\textsuperscript{31} project in the context of Semantic Business Process Modelling - SBPM - (Hepp et al., 2005). This emergent research area proposes the use of ontologies and Semantic Web Services in order to provide an unified view of business processes in a machine understandable way. One of the concerns of Business Process Management (BPM) is to provide process modelling languages and tools to facilitate bridging between the business and Information Technology (IT). However, one major obstacle to the complete realization of BPM is that the business process space inside the organisation,

\textsuperscript{30} http://www.bpmn.org
\textsuperscript{31} Semantics Utilised for Process Management within and between Enterprises (http://www.ip-super.org/)
from the business expert perspective to the actual implementation is widely not accessible at the semantic level and thus neither to machine reasoning.

Within the SUPER project, we provide a set of integrated ontologies developed in WSML for solving the problem above. In particular, we provide ontologies for a number of popular standards (BPMN, EPC, BPEL) and the novel Business Process Modelling Ontology (BPMO), which we describe in this Chapter. The goal is to support a number of BPM life-cycle activities at the semantic level, including modelling, querying, mediation, discovery, execution and monitoring. One major task within this approach is thus handling the translations between the provided ontologies in order to navigate from different views at the business level to the IT view at the execution level.

This Chapter describes BPMO in detail and our model-driven approach and implemented translator for transforming instances of BPMO to instances of an ontology of BPEL. Appendix E contains the complete specification of BPMO; and appendix F contains details of the implemented translator.

With respect to BPMO, our contribution consists of the overall design and modelling of a number of elements including the concept (and attributes) BusinessActivity; its sub-concepts including Process, Task, GoalTask, Send, Receive, MediationTask; plus the concept SemanticCapability. We also defined the concept Workflow and the division between block-patterns and graph-patterns. With respect to the

---

32 http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
BPMO2SBPEL translator, we contributed with the whole implementation including the translation rules and Java API (as described later in this Chapter).

5.2 Approach

As mentioned in the introduction, the Business Process Modelling Ontology (BPMO) is part of an approach to modelling business processes at the semantic level, integrating knowledge about the organisational context, workflow activities and executable services (see Figure 5-1). BPMO provides support for various BPM activities, abstracting from specific notations in a way which crosses domains and organisational boundaries. BPMO harness a number of knowledge representation and reasoning techniques so that business process workflows can: a) be exposed and shared through semantic descriptions; b) refer to semantically annotated business data and services; c) incorporate heterogeneous data though semantic mappings; and d) be queried using a reasoner or inference engine. We argue therefore, that BPMO enables seamless interoperation, querying, sharing, mediation and translation of business processes.

![Figure 5-1 BPMO Model Main Elements](image)

There are various reasons that motivate the creation of BPMO. First, BPMO provides a bridge between business processes defined by business analysts (e.g. BPMN) and
business processes defined by developers for execution (as seen in languages used for process execution). Second, BPMO provides links to organisational aspects that can be modelled independently for different domains. Third, BPMO can be used during the whole life cycle of Business Process Management, from modelling, to execution, to analyses, and monitoring. Finally, BPMO is the primary ontology intended to be used in a top-down approach to modelling processes at the semantic level using a modelling tool. As a result business analysts can use BPMO to model and query processes before configuring the process for execution.

As we will show in the next section, the BPMO model captures domain independent organisational aspects, control-flow features of notations such as BPMN, via a number of workflow patterns as in (Aaslt et al., 2003); process interaction features from BPEL; and finally service description and invocation features from Semantic Web Services (SWS). BPMO builds on the formalization of Business Process Diagrams as presented in (Ouyang et al., 2006), and as such is oriented towards the production of well-formed workflow models, where graphs decompose unambiguously into sub-graphs that start and end with compatible constructs.

A BPMO diagram can be generated using the WSMO Studio BPMO Modeller tool\(^{33}\), which automatically generates instances of BPMO in WSML. More precisely, we use WSML-Flight, which adds F-Logic like features to the WSML core, directly supporting WSMO Web Service descriptions. WSML-Flight also allows us to apply

\(^{33}\)http://www.wsmostudio.org/
data mappings (via rule-type axioms) directly in the ontology language without having to rely on a hybrid approach with a separate rule language. It is also possible for business analysts to create alternative organisational ontologies to define BPMO process organisational attributes.

In addition, as our second contribution in this chapter, we provide a translator from BPMO to sBPEL. In Figure 5-2 we depict a subset of the ontologies (rounded rectangles) available within SUPER for the purpose of explaining the context of our translation approach. The main ontology is BPMO (see Section 5.3) to and from which corresponding translations are performed (large arrows in the picture). BPMO imports UPO (Upper-level Process Ontology), an umbrella ontology defining generic business process concepts, shared by all ontologies in SUPER. sEPC and sBPMN are the two ontologies created to semantically annotate the corresponding standard notations used to model process workflows at the business level. SBPEL (see Section 5.4.1) is an ontology for the BPEL language, which is used by IT experts to execute process workflows. These ontologies can be grounded to any tool-specific syntactic...
format of the respective notation (rectangles in the picture) via straightforward serializations.

BPMO and the related ontologies mentioned above are publicly available at the SUPER website (http://www.ip-super.org). In the following we briefly describe the ontologies in Figure 5-2.

*UPO (Upper-level Process Ontology)* defines generic notions of business processes that can be used and refined across all other SUPER ontologies.

*sEPC (Semantic EPC)* is an ontology for the EPC notation. An sEPC model represents a business process as a set of flows between organisational functions and events. The sEPC ontology main concepts include *Function, Event, Logical Connector* and *Process Hierarchy*.

*sBPMN (Semantic BPMN)* is an ontology for the BPMN notation. BPMN is a graphical notation which represents a business process through a control-flow. It also represents the messages between partner processes. sBPMN includes constructs such as *Process, Event, Message Flow and Sequence Flow*.

*sBPEL (Semantic BPEL)* is an ontology of BPEL with extended constructs. This ontology can be serialized into a XML format – BPEL4SWS – that contain semantic extensions to BPEL, so that processes can automatically execute tasks as Semantic Web Services.
5.3 BPMO Description

BPMO\textsuperscript{34} is a representation for high-level business process workflow models, abstracting from existing business process notations. Nevertheless, the workflow elements of a BPMO process comply with a corresponding subset of BPMN control-flow elements and are informed by, and named according to Workflow Patterns (Aalst et al., 2003). Moreover, BPMO concepts related to interaction activities (e.g. Send, Receive) have a number of attributes that correspond to BPEL constructs. See Table 5-1 and Table 5-2 for the description of BPMO constructs and mappings to BPMN and BPEL.

Basically, a BPMO process description captures the business context of the modelled process and contains the process workflow, which represents the behaviour of the process (through control-flow and data-flow constructs) and process activities (through Tasks). The main BPMO process elements are structured as follows:

- **Workflow** – The business process container for Workflow Elements. The initial Workflow Element is a Start event or a block pattern, commonly a Sequence or ParallelSplit Synchronise. If the Start event is present, subsequent elements will be linked in graph style by Controlflow Connectors. If the Workflow Element is a Sequence, a sequence flow is implicit between the contained elements. If the Workflow Element is ParallelSplitSynchronise, a parallel flow is implicit.

\textsuperscript{34}http://ip-super.org/ontologies/process/bpmo/v2.0.1#bpmo
- **Workflow Elements** – These are general elements that belong to a business process workflow, including *Processes, Tasks, Events*, block patterns and graph patterns;

- **Block Patterns** – These are structured or pattern-based control-flow elements representing workflow decision points (gateways), including *Sequence, ParallelSplit, Synchronise, ExclusiveChoiceMerge, DeferredChoiceMerge, While, Repeat*, and so on.

- **Graph patterns** – These are link based control-flow elements representing workflow decision points (gateways), including *ParallelSplit, ExclusiveChoice, DeferredChoice, SimpleMerge, Synchronise*, and so on.

As can be seen, BPMO combines features of block-oriented and graph-oriented workflow models. The main purpose of block patterns is to explicitly represent structured elements and workflow patterns that can be used to facilitate process verification and the translation to notations in the execution level.

**Table 5-1** Description of BPMO elements and correspondence to BPMN

<table>
<thead>
<tr>
<th>BPMN Element</th>
<th>BPMO Element/Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process/SubProcess</td>
<td>Process/Subprocess</td>
<td>A business process</td>
</tr>
<tr>
<td>Start Event</td>
<td>Start Event</td>
<td>An event signaling the start of a process</td>
</tr>
<tr>
<td>Intermediate Message Event</td>
<td>Receive Message Event</td>
<td>An event signaling that a message has arrived. Provides a semantic description (SWS) of the recipient.</td>
</tr>
<tr>
<td>Intermediate Timer Event</td>
<td>Timer</td>
<td>An event signaling that a specific time has been reached</td>
</tr>
<tr>
<td>BPMN Element</td>
<td>BPMO Element/Concept</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>--</td>
<td>Error</td>
<td>An event signaling that an error has occurred</td>
</tr>
<tr>
<td>End Event</td>
<td>End Event</td>
<td>An event signaling the end of a process</td>
</tr>
<tr>
<td>Task</td>
<td>Task</td>
<td>An atomic activity within a Process.</td>
</tr>
<tr>
<td>--</td>
<td>Goal Task</td>
<td>A Task with an attached Semantic Capability. Used for invoking SWS goals</td>
</tr>
<tr>
<td>--</td>
<td>Send</td>
<td>A Task for sending messages. Provides a semantic description of the requested capability.</td>
</tr>
<tr>
<td>--</td>
<td>Receive</td>
<td>A Task for receiving messages. Provides a semantic description of the provided capability.</td>
</tr>
<tr>
<td>Parallel Fork Gateway</td>
<td>ParallelSplit</td>
<td>A gateway (or decision point) for creating concurrent branches</td>
</tr>
<tr>
<td>Parallel Join Gateway</td>
<td>Synchronization</td>
<td>A gateway for synchronizing concurrent branches</td>
</tr>
<tr>
<td>--</td>
<td>ParallelSplitSynchronize</td>
<td>Parallel Split with an implicit Synchronization - Block pattern</td>
</tr>
<tr>
<td>XOR Decision Gateway</td>
<td>ExclusiveChoice</td>
<td>A decision gateway for selecting one out of a set of mutually exclusive alternative branches based on data. One of the branches may be default.</td>
</tr>
<tr>
<td>--</td>
<td>ExclusiveChoiceMerge</td>
<td>Exclusive Choice with an implicit Simple Merge - Block pattern</td>
</tr>
<tr>
<td>Event-based XOR Decision Gateway</td>
<td>DeferredChoice</td>
<td>A decision gateway for selecting one out of a set of mutually exclusive alternative branches based on external event</td>
</tr>
<tr>
<td>--</td>
<td>DeferredChoiceMerge</td>
<td>Deferred Choice with an implicit Simple Merge - Block pattern</td>
</tr>
<tr>
<td>XOR Merge Gateway</td>
<td>SimpleMerge</td>
<td>Gateway for joining a set of mutually exclusive alternative branches into one branch</td>
</tr>
<tr>
<td>OR Decision Gateway</td>
<td>MultipleChoice</td>
<td>A decision gateway for selecting many out of a set of alternative branches into several parallel branches based on data. One of the branches may be default.</td>
</tr>
<tr>
<td>OR Merge Gateway</td>
<td>MultipleMerge</td>
<td>unsynchronised convergence of two</td>
</tr>
</tbody>
</table>
We will discuss next the use of a number of key BPMO concepts, including *Process*, *Business Activity*, *Task* (*Send*, *Receive*, *GoalTask*), *SemanticCapability*, *MediationTask* and *DataMediator*.

The *Process* concept (shown in Listing 5-1) defines several organisational attributes, by inheriting from *BusinessActivity*, according to the types *BusinessDomain*, *BusinessFunction*, *BusinessStrategy*, *BusinessPolicy*, *BusinessProcessMetrics*,

<table>
<thead>
<tr>
<th>BPMN Element</th>
<th>BPMO Element/Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>MultipleChoiceMerge</td>
<td>Multiple Choice with an implicit Multiple Merge – Block pattern</td>
</tr>
<tr>
<td>Synchronizing Merge</td>
<td>MultipleMergeSynchronize</td>
<td>Synchronised convergence of two or more distinct branches</td>
</tr>
<tr>
<td>Interleaved Parallel Routing</td>
<td>InterleavedParallelRouting</td>
<td>A decision gateway for selecting many out of a set of alternative branches into several parallel branches based on external events</td>
</tr>
<tr>
<td>Discriminator</td>
<td>Discriminator</td>
<td>Convergence of two or more branches such that the first activation of an incoming branch results in the subsequent activity being triggered and subsequent activations of remaining incoming branches are ignored.</td>
</tr>
<tr>
<td>Structured cycle</td>
<td>Repeat</td>
<td>A structured loop where the condition is evaluated after the body of the loop is executed</td>
</tr>
<tr>
<td>Structured cycle</td>
<td>While</td>
<td>A structured loop where the condition is evaluated before the body of the loop is executed</td>
</tr>
<tr>
<td>Sequence Flow</td>
<td>(Implicit Sequence Flow or explicit ControlFlow Connector concept)</td>
<td>Links two Workflow Elements</td>
</tr>
<tr>
<td>--</td>
<td>Sequence</td>
<td>An ordered set of activities with an implicit Sequence Flow – Block pattern</td>
</tr>
<tr>
<td>Message Flow</td>
<td>(there is no explicit concept. It is captured by <em>MessageFrom</em> and <em>MessageTo</em> attributes)</td>
<td>Links a message sender to a message receiver and shows the data</td>
</tr>
</tbody>
</table>
BusinessProcessGoal and BusinessResource. These business-level concepts (attribute types) are primarily defined in external ontologies, which model a specific business domain and organisation. These ontologies are linked to the BPMO process by subclassing the UPO concept (note that upo# is the prefix for the UPO namespace).

As a result, we enable the querying of processes against organisational aspects by business analysts (see example in the next Chapter). The Process itself can also have a corresponding Web Service description (hasWSDescription attribute). In addition, the Process concept defines the process workflow (attribute hasWorkflow). The concept Workflow defines the first element of the workflow (hasFirstWorkflowElement). The workflow is modelled with Workflow Elements (see above) following the first element.

**Listing 5-1. Business Activity and Process Concepts in BPMO**

```plaintext
Concept BusinessActivity subConceptOf upo#BusinessActivity
  hasName ofType (0 1) _string
  hasDescription ofType (0 1) _string
  hasNonFunctionalProperties ofType (0 1) BusinessActivityNonFunctionalProperties
  hasBusinessDomain ofType upo#BusinessDomain
  hasBusinessFunction ofType upo#BusinessFunction
  hasBusinessStrategy ofType upo#BusinessStrategy
  hasBusinessPolicy ofType upo#BusinessPolicy
  hasBusinessProcessMetrics ofType upo#BusinessProcessMetrics
  hasBusinessProcessGoal ofType upo#BusinessProcessGoal
  hasBusinessResource ofType upo#Resource

concept Process subConceptOf {BusinessActivity, upo#BusinessProcessModel}
  hasWSDescription ofType (0 1) SemanticCapability
  hasWorkflow ofType (0 1) Workflow

concept Workflow subConceptOf upo#ProcessOrchestrationSpecification
  hasHomeProcess ofType (0 1) Process
  hasFirstWorkflowElement ofType (1 1) WorkflowElement
```

The concepts related to interaction tasks in a process are GoalTask, Receive, Send and ReceiveMessageEvent (Listing 5-2), which are subconcepts of Task (see more details in Table 5-2). A Task is also a Business Activity (as in Listing 5-1). Tasks have attributes to represent information about the interaction with a partner process, such as
partner role (hasPartnerRole), inputs (hasInputDescription) and outputs (hasOutputDescription). Most attribute types in Tasks are defined as SemanticCapability which is a wrapper for ontology elements or service descriptions.

For example, a SemanticCapability instance can refer to the URI of an concept within an ontology or to the URI of a Web Service or Goal description.

Listing 5-2 Concepts Related to Interaction Tasks in BPMO

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoalTask</td>
<td>Represents an atomic activity, which can be automatically achieved through a SWS invocation (synchronous communication). The attribute hasPartnerGoal is used in this case to refer to the Goal description. The hasInputDescription and hasOutputDescription attributes refer to the semantic descriptions of request and response data respectively. The requestsCapability and providesCapability attributes refer to the semantic descriptions of operations related to request and response respectively. The Send and Receive tasks are similar to Goal tasks, but they are used for asynchronous communication. A Receive task can be associated with a Send in the...</td>
</tr>
</tbody>
</table>

A GoalTask represents an atomic activity, which can be automatically achieved through a SWS invocation (synchronous communication). The attribute hasPartnerGoal is used in this case to refer to the Goal description. The hasInputDescription and hasOutputDescription attributes refer to the semantic descriptions of request and response data respectively. The requestsCapability and providesCapability attributes refer to the semantic descriptions of operations related to request and response respectively. The Send and Receive tasks are similar to Goal tasks, but they are used for asynchronous communication. A Receive task can be associated with a Send in the...
same workflow via the \textit{hasSendCounterpart} attribute (and conversely for \textit{Send}).

\textit{ReceiveMessageEvent} works as a \textit{Receive} task, but is also associated to an event, which is triggered when a message is received.

\textbf{Listing 5-3} Concepts related to Mediation in BPMO

\begin{verbatim}
concept MediationTask subConceptOf Task
  hasSourceTask ofType (0 1) Task
  hasTargetTask ofType (0 1) Task
  hasDataMediator ofType DataMediator

concept Mediator subConceptOf upo#BusinessProcessMediator
  hasName ofType (0 1) _string
  hasDescription ofType (0 1) _string

concept ProcessMediator subConceptOf Mediator
  hasSourceProcess ofType (1 1) Process
  hasTargetProcess ofType (1 *) Process
  hasMediationProcess ofType (0 1) MediationProcess
  hasSWSMediator ofType (0 1) SemanticCapability

concept DataMediator subConceptOf Mediator
  hasMediator ofType SemanticCapability
  hasMediationService ofType SemanticCapability
  hasInputDescription ofType (0 1) SemanticCapability
  hasOutputDescription ofType (0 1) SemanticCapability

concept MediationProcess subConceptOf Process
\end{verbatim}

The concepts related to data and process Mediation are shown in Listing 5-3. See also the examples in Chapter 6 (Section 6.3 and Section 6.4). A \textit{MediationTask} is a task that provides data mapping specifications to be used between tasks during runtime. A \textit{MediationTask} can have one or more \textit{DataMediators}. The \textit{DataMediator} concept refers to a data mediator or mediation service and the input and output for them. In a typical use case, the \textit{hasMediator} attribute will refer to a mapping definition (URI), the \textit{hasInputDescription} will refer to a source ontology (URI) and the \textit{hasOutputDescription} will refer to a target ontology (URI). In addition, the \textit{ProcessMediator} concept is used as a descriptor to identify a process with a mediation role (\textit{hasMediationProcess}) and mediated processes (\textit{hasSourceProcess}, \textit{hasTargetProcess}) in order to facilitate the job of tools for verification and creation of
mediation processes. The ProcessMediator can also refer to a mediator component (hasSWSMediator).

### 5.4 A Translator for BPMO

In this section we describe our model-driven approach and implemented translator for transforming instances of BPMO to instances of sBPEL. More precisely, we implement mappings using ATL\(^{35}\) rules, using the XML format of WSML as both source and target models of the ATL transformation engine. The result of the translation is a portable file containing a semantically annotated executable business process model.

It is out of the scope of this thesis to explain sBPEL in detail, but the translation is a necessary exercise in fulfilling the purpose of BPMO as a bridging ontology. In fact, the translation gives feedback into the correctness of the BPMO model itself. The translator developed here is a preliminary prototype, which shows the viability of using BPMO to bridge the gap between business-level processes and executable processes.

Next, we give an overview of sBPEL, including the mappings between BPMO and sBPEL; present the translation approach we adopted; provide the specification of the translation rules; and the implementation of the translator. The next chapter shows the results of the translator through a use case.

---

\(^{35}\) [http://www.eclipse.org/M2M/ATL](http://www.eclipse.org/M2M/ATL)
5.4.1 Mapping BPMO to Semantic BPEL

Semantic BPEL (sBPEL)\textsuperscript{36} is an ontology for BPEL4SWS (Nitzsche et al., 2007), which is an extension of BPEL4WS\textsuperscript{37} (BPEL for short), a language for specifying the composition of Web Services using a control-flow based approach. Control-flow constructs are either structured activities (e.g. sequence, flow, if), or basic activities including assign and interaction activities (e.g. receive, reply, invoke, pick). In addition, links (dependencies) can be added between activities within a flow (parallel execution) activity in a graph-based style. BPEL4SWS uses the extensibility elements of BPEL in order to add semantic annotations of data and services to the process. It also defines the Interaction extension, which can be used to group a number of interaction activities into a Conversation, for modelling long running conversational interaction among partners. The sBPEL ontology is provided in Appendix E (Section A3) for convenience.

For example, as shown in Listing 5-4, the Receive concept (sub-concept of Interaction and NewActivityType) represents one of the interaction activities of BPEL4SWS in sBPEL. The Receive activity can be added to the control-flow via the ExtensionActivity concept (hasActivity attribute). Receive must belong to a Conversation (belongsToConversation attribute), which in turn must define an interface (describesInterface attribute) that has a Semantic Web Service description.

\textsuperscript{36} http://ip-super.org/ontologies/process/sbpel/v2.0.0#sbpel
\textsuperscript{37} http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
(hasWebServiceDescription attribute). Note also that Receive (hasVariable attribute) uses SemanticVariable (hasSemanticType attribute) to annotate the data received.

Listing 5-4 Example of concepts in sBPEL

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemanticProcess</td>
<td>subConceptOf bpel#Process</td>
</tr>
<tr>
<td></td>
<td>hasConversation ofType Conversation</td>
</tr>
<tr>
<td></td>
<td>hasPartner ofType Partner</td>
</tr>
<tr>
<td></td>
<td>hasSemanticOnMessage ofType SemanticOnMessage</td>
</tr>
<tr>
<td>Receive</td>
<td>subConceptOf [bpel#Interaction, bpel#NewActivityType]</td>
</tr>
<tr>
<td></td>
<td>doesCreateInstance ofType [0 1] _boolean</td>
</tr>
<tr>
<td></td>
<td>belongsToConversation ofType [1] Conversation</td>
</tr>
<tr>
<td></td>
<td>hasVariable ofType [1] SemanticVariable</td>
</tr>
<tr>
<td>Conversation</td>
<td>hasName ofType [1] _string</td>
</tr>
<tr>
<td></td>
<td>describesInterface ofType [1] InterfaceDescription</td>
</tr>
<tr>
<td></td>
<td>correspondsTo ofType bpmo#Process</td>
</tr>
<tr>
<td>IncomingInterface</td>
<td>subConceptOf InterfaceDescription</td>
</tr>
<tr>
<td></td>
<td>hasWebServiceDescription ofType [1] _string</td>
</tr>
<tr>
<td>SemanticVariable</td>
<td>subConceptOf bpel#Variable</td>
</tr>
<tr>
<td></td>
<td>hasSemanticType ofType [1] _string</td>
</tr>
<tr>
<td>Partner</td>
<td>hasName ofType [1] _string</td>
</tr>
<tr>
<td></td>
<td>hasBusinessEntity ofType [0 1] _string</td>
</tr>
<tr>
<td></td>
<td>hasConversation ofType [1 *] Conversation</td>
</tr>
<tr>
<td>ExtensionActivity</td>
<td>subConceptOf bpel#BasicActivity</td>
</tr>
<tr>
<td></td>
<td>hasActivity ofType [1] NewActivityType</td>
</tr>
</tbody>
</table>

Table 5-2 shows the mappings from sBPEL attributes to the applicable subset of BPMO elements. Note that some of the attributes shown in this table are inherited from parent concepts (super classes). We do not include in this table the business related attributes of BPMO Process and Task since they are not translated to sBPEL.

Table 5-2 Description of BPMO elements and the mappings to sBPEL elements

<table>
<thead>
<tr>
<th>sBPEL Concept/Attribute</th>
<th>BPMO Concept/Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemanticProcess</td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td>- hasWSdescription</td>
<td>Web Service (SWS) description of this process.</td>
</tr>
<tr>
<td>- hasActivity</td>
<td>- hasWorkflow (via WorkflowElement)</td>
<td>The container for this process’ workflow</td>
</tr>
<tr>
<td>sBPEL Concept/Attribute</td>
<td>BPMO Concept/Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>--</td>
<td>-hasInvolvedRole</td>
<td>Description of this process’s role</td>
</tr>
<tr>
<td>- hasPartner</td>
<td></td>
<td>(Generated from BusinessRoles)</td>
</tr>
<tr>
<td>- hasConversation</td>
<td></td>
<td>(Generated from Tasks)</td>
</tr>
<tr>
<td>- hasSemantic OnMessage</td>
<td></td>
<td>(Generated from ReceiveMessage Event)</td>
</tr>
<tr>
<td>- hasVariable</td>
<td></td>
<td>(Generated from Task’s inputs and outputs)</td>
</tr>
<tr>
<td>Empty</td>
<td>SubProcess</td>
<td>A process within a process.</td>
</tr>
<tr>
<td>Empty</td>
<td>Task</td>
<td>The super concept for GoalTask, Send, Receive and MediationTask. An instance of this super concept can also represent a manual task or an undefined activity.</td>
</tr>
<tr>
<td>SendReceive</td>
<td>GoalTask</td>
<td>(Generates Outgoing Interface)</td>
</tr>
<tr>
<td>+ Conversation</td>
<td></td>
<td>(via Semantic Capability)</td>
</tr>
<tr>
<td>- hasPartnerGoal</td>
<td>-hasPartnerRole</td>
<td>Semantic description of the partner’s process SWS Goal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(via Semantic Capability)</td>
</tr>
<tr>
<td>- hasPartnerRole</td>
<td></td>
<td>Description of the role of the partner including the organisation</td>
</tr>
<tr>
<td>- hasInputVariable</td>
<td>-hasInputDescription</td>
<td>Semantic description of input data</td>
</tr>
<tr>
<td></td>
<td>(via Semantic Capability)</td>
<td></td>
</tr>
<tr>
<td>- hasOutputVariable</td>
<td>-hasOutputDescription</td>
<td>Semantic description of output data</td>
</tr>
<tr>
<td></td>
<td>(via Semantic Capability)</td>
<td></td>
</tr>
<tr>
<td>- messageTo</td>
<td></td>
<td>Receiving Task/Event in partner’s process</td>
</tr>
<tr>
<td>- messageFrom</td>
<td></td>
<td>Sending Task/Event in partner’s process</td>
</tr>
<tr>
<td>- requestsCapability</td>
<td></td>
<td>Semantic description of out operation</td>
</tr>
<tr>
<td>sBPEL Concept/Attribute</td>
<td>BPMO Concept/Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>--</td>
<td>- providesCapability</td>
<td>Semantic description of in operation</td>
</tr>
<tr>
<td>Send (via Extension Activity)</td>
<td>Send</td>
<td>A Task for sending data</td>
</tr>
<tr>
<td>(Generates Outgoing Interface)</td>
<td>- hasPartnerWebService (via Semantic Capability)</td>
<td>Web Service (SWS) description of partner’s process</td>
</tr>
<tr>
<td>--</td>
<td>- hasPartnerRole</td>
<td>Description of the partner’s business role</td>
</tr>
<tr>
<td>--</td>
<td>- hasReceiveCounterpart (supports in identifying interactions)</td>
<td>Receive Task/Event in this process, which is the counterpart of this Send.</td>
</tr>
<tr>
<td>--</td>
<td>- messageTo</td>
<td>Receiving Task/Event in partner’s process</td>
</tr>
<tr>
<td>- hasVariable</td>
<td>- hasOutputDescription (via Semantic Capability)</td>
<td>Semantic description of output data</td>
</tr>
<tr>
<td>--</td>
<td>- requestsCapability</td>
<td>Semantic description of out operation</td>
</tr>
<tr>
<td>Receive (via Extension Activity) + Conversation</td>
<td>Receive</td>
<td>A Task for receiving data</td>
</tr>
<tr>
<td>(Generates Incoming Interface)</td>
<td>- hasPartnerWebService (via Semantic Capability)</td>
<td>Web Service (SWS) description of partner’s process</td>
</tr>
<tr>
<td>(Generates Partner)</td>
<td>- hasPartnerRole</td>
<td>Description of the partner’s business role</td>
</tr>
<tr>
<td>--</td>
<td>- hasSendCounterpart (supports in identifying interactions)</td>
<td>Send Tasks in this process, which is the counterpart of this Receive.</td>
</tr>
<tr>
<td>--</td>
<td>- messageFrom</td>
<td>Sending Task in partner’s process</td>
</tr>
<tr>
<td>--</td>
<td>- providesCapability</td>
<td>Semantic description of in operation</td>
</tr>
<tr>
<td>- hasVariable</td>
<td>- hasInputDescription (via Semantic Capability)</td>
<td>Semantic description of input data</td>
</tr>
<tr>
<td>SemanticOnMessage</td>
<td>ReceiveMessageEvent (same attributes as Receive)</td>
<td>An Event signaling the receipt of data. (Also subclass of Receive)</td>
</tr>
<tr>
<td>Assign</td>
<td>MediationTask</td>
<td>A Task for data mediation</td>
</tr>
<tr>
<td>- hasAssignOperation (via Extension AssignOperation)</td>
<td>- hasDataMediator</td>
<td>Description of a SWS data mediator</td>
</tr>
</tbody>
</table>
### Chapter 5 Mediation and Business Process Modelling

<table>
<thead>
<tr>
<th>aBPEL Concept/Attribute</th>
<th>BPMO Concept/Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>- hasSourceTask</td>
<td>The source task of this mediation</td>
</tr>
<tr>
<td>--</td>
<td>- hasTargetTask</td>
<td>The target task of this mediation</td>
</tr>
<tr>
<td>Mediate</td>
<td>DataMediator</td>
<td>Description of a SWS data mediator</td>
</tr>
<tr>
<td>- usesDataMediator (via DataMediator)</td>
<td>- hasMediator (via Semantic Capability)</td>
<td>Link (URI) to a SWS ontology mediator</td>
</tr>
<tr>
<td>- usesDataMediator (via DataMediator)</td>
<td>- hasMediationService (via Semantic Capability)</td>
<td>Link (URI) to a SWS Goal or Web Service</td>
</tr>
<tr>
<td>- hasInputVariable</td>
<td>- hasInputDescription (via Semantic Capability)</td>
<td>Link (URI) to semantically described input data (ontology)</td>
</tr>
<tr>
<td>- hasOutputVariable</td>
<td>- hasOutputDescription (via Semantic Capability)</td>
<td>Link (URI) to semantically described output data (ontology)</td>
</tr>
<tr>
<td>Sequence</td>
<td>Sequence</td>
<td>Container of sequential elements showing order.</td>
</tr>
<tr>
<td>- hasOrderedActivity</td>
<td>- hasOrderedElement</td>
<td>Container of sequential elements showing order.</td>
</tr>
<tr>
<td>If</td>
<td>ExclusiveChoice/Merge</td>
<td>Points to several conditional branches. Translation will reduce composed branches to a sequence</td>
</tr>
<tr>
<td>- hasIfBranch/hasElseIfBranch/</td>
<td>- hasConditionalBranch</td>
<td>Points to several conditional branches. Translation will reduce composed branches to a sequence</td>
</tr>
<tr>
<td>hasElseBranch</td>
<td>- hasDefaultBranch</td>
<td>Default branch</td>
</tr>
<tr>
<td>While</td>
<td>While</td>
<td>Points to a conditional expression</td>
</tr>
<tr>
<td>- hasCondition</td>
<td>- hasCondition</td>
<td>Points to a conditional expression</td>
</tr>
<tr>
<td>- hasActivity</td>
<td>- executes</td>
<td>Points to a body of activities. Translation will reduce a composed body to a sequence</td>
</tr>
<tr>
<td>RepeatUntil</td>
<td>Repeat</td>
<td>Points to a conditional expression</td>
</tr>
<tr>
<td>- hasCondition</td>
<td>- hasCondition</td>
<td>Points to a conditional expression</td>
</tr>
<tr>
<td>- hasActivity</td>
<td>- executes</td>
<td>Points to a body of activities. Translation will reduce composed branches to a sequence</td>
</tr>
<tr>
<td>SemanticPick (via Extension Activity)</td>
<td>DeferredChoice/Merge</td>
<td>Points to several branches of message events or a timer events. Translation will reduce composed branches to a sequence</td>
</tr>
<tr>
<td>- hasOnMessage/hasOnAlarm</td>
<td>- hasEventBranch</td>
<td>Points to several branches of message events or a timer events. Translation will reduce composed branches to a sequence</td>
</tr>
</tbody>
</table>
## CHAPTER 5 MEDIATION AND BUSINESS PROCESS MODELLING

<table>
<thead>
<tr>
<th>sBPEL Concept/Attribute</th>
<th>BPMO Concept/Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow</td>
<td>ParallelSplitSynchronize</td>
<td>Points to several parallel branches. Translation will reduce composed branches to a sequence</td>
</tr>
<tr>
<td>- hasActivity</td>
<td>- hasBranch</td>
<td>Points to several parallel branches. Translation will reduce composed branches to a sequence</td>
</tr>
<tr>
<td>Flow + Links</td>
<td>MultipleChoice/Merge</td>
<td></td>
</tr>
<tr>
<td>- hasActivity</td>
<td>- hasBranch</td>
<td>Points to several parallel branches. Translation will reduce composed branches to a sequence</td>
</tr>
<tr>
<td>Partner</td>
<td>BusinessRole</td>
<td>Description of a business role usually associated with a process or partner process.</td>
</tr>
<tr>
<td>- hasName</td>
<td>- hasName</td>
<td>Role name</td>
</tr>
<tr>
<td></td>
<td>- hasDescription</td>
<td>Role description</td>
</tr>
<tr>
<td>- hasBusinessEntity (only name of Organisation)</td>
<td>- hasOrganisation (via upo#Organisation)</td>
<td>Organisation associated with this role.</td>
</tr>
<tr>
<td>- hasConversation (Generated from the enclosing Task)</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>SemanticCapability</td>
<td></td>
<td>Provides a pointer (URI) to a Concept Description</td>
</tr>
<tr>
<td></td>
<td>- hasSemanticDescription</td>
<td>URI</td>
</tr>
</tbody>
</table>

In summary, the main target sBPEL elements/parent (bpel) into which BPMO elements will be automatically translated are:

- **SemanticProcess/Process (Process),**
- **SendReceive/Interaction/NewActivityType (Invoke),**
- **Receive/Interaction/NewActivityType (Receive),**
- **Send/Interaction/NewActivityType (Reply),**
- **Mediate/ExtensionAssignOperation (Copy),**
A translation from BPMO to sBPEL can be based on structured or graph elements. The translation of a structured BPMO process will be from a BPMO diagram, which contains elements corresponding to the above list only, with the restriction that the branches contain one single block-pattern. That is, the BPMO diagram uses block-patterns and resemble the structured ones draw with BPEL tools. In this case the instances of BPMO and the translated instances of sBPEL are structured.

A graph-based translation will be from a BPMO diagram, which contains elements corresponding to the above list only with no restriction for composed branches using block and graph patterns. In this case, the BPMO instance will contain explicit Control flow Connectors. The approach used in this case is to detect the composed branches in BPMO and translate into sequences in sBPEL.
5.4.2 Translation Approach and Implementation

In this section we describe the implementation of our BPMO2SBPEL translator, which transforms between instances of a BPMO model and a sBPEL model. The translator takes as input a WSML file containing BPMO instances of an individual business process and generates as output a WSML file containing corresponding sBPEL instances (see also the use case in next Chapter). The generated sBPEL file can be used in a later stage for serialization and then execution in an appropriate engine.

We have developed a standalone java API for BPMO2SBPEL (available at http://kmi.open.ac.uk/projects/super/BPMO2SBPEL-2.0.zip). This API uses the WSMO4J API (http://wsmo4j.sourceforge.net) to parse and serialize XML versions of WSML as required; creates the input files from the XML files; and launches the ATL engine. The source code of the java API is given in Appendix F (Section F.2).

The translator has been developed using EMF\(^{38}\) (Eclipse Modelling Framework). The translator rules are written in ATL\(^{39}\) (Atlas Transformation Language), which is a hybrid language (a mix of declarative and imperative constructors), designed to express model transformations. As illustrated in the diagram of Figure 5-3, the ATL engine requires meta-models in the ECORE format and conforming XMI source and target models for the transformation. Since our source and target models are in

\(^{38}\) http://www.eclipse.org/modelling/emf
\(^{39}\) http://www.eclipse.org/M2M/ATL
WSML, we create a WSML meta-model using a specific EMF tool, which takes a XSD file. We use the XML syntax of WSML\textsuperscript{40} in order to generate the meta-model (Wsml.ecore) from WSML XSD. In addition, the input WSML XMI instances are generated (programmatically) from given WSML XML instances.

\begin{figure}[h]
\centering
\begin{tikzpicture}
  \node (wsmlxsd) at (0,0) {WSML XSD};
  \node (emf) at (2,0) {EMF};
  \node (wsml.ecore) at (2,2) {WSML ECORE};
  \node (wsmlxmi.bpmo.instance) at (0,2) {WSML XMI BPMO instance};
  \node (wsmlxmi.wsml.instance) at (2,2) {WSML XMI sBPEL instance};
  \draw [-latex] (wsmlxsd) -- (emf);
  \draw [dashed] (emf) -- (wsml.ecore);
  \draw [dashed] (wsmlxmi.bpmo.instance) -- (emf);
  \draw [dashed] (wsmlxmi.wsml.instance) -- (atl)
  \node (atl) at (2,0) {ATL};
\end{tikzpicture}
\caption{ATL Transformation of WSML instances}
\end{figure}

There is a small issue with mixed XSD types within WSML XSD, which cannot be processed by ATL. The work around this problem was to create another version of the WSML XSD and corresponding meta-model (syntax.ecore) using strings for primitive types instead of mixed ones, and use both ECORE source models for the translation.

The complete specification of the ATL rules which provides the mappings for our translator is available within the distribution package together with the API mentioned previously. The complete specification of the rules is also listed in Appendix F.

The syntax of a basic ATL rule (see guide\textsuperscript{41}) is shown below.

\begin{verbatim}
rule name { from src: type (filter) to [tgt: type (body)]+}
\end{verbatim}

\textsuperscript{40} http://www.wsmo.org/TR/d16/d16.1/v0.21/xml-syntax/wsml-xml-syntax.xsd
\textsuperscript{41} http://wiki.eclipse.org/ATL/User_Guide
From the format above, rule has a name; from provides the source pattern; and to provides the target pattern (the plus sign indicates more than one occurrence of the enclosed pattern). src defines the source pattern element and tgt defines the target pattern element according to schema types in the respective models. A rule is triggered when filter (a conditional statement) holds true for the matched source model element, generating one or more target model elements, according to the operations in body. Within body, features of the source element can be transformed into features of the target element using several set-based operations from ATL. The reader is referred to the Eclipse website mentioned before for details of the ATL language.

```xml
<instance name='http://ip-super.org/examples/process/bpmo/v2.0.1/examples#Receive_ContentRequest'>
  <memberOf>http://ip-super.org/ontologies/process/bpmo/v2.0.1#Receive</memberOf>
  - <attributeValue name='http://ip-super.org/ontologies/process/bpmo/v2.0.1#hasName'>
    <value type='http://www.wsmo.org/wsml/wsml-syntax#string'>Receive Content Request</value>
  </attributeValue>
  - <attributeValue name='http://ip-super.org/ontologies/process/bpmo/v2.0.1#hasHomeProcess'>
    <value type='http://ip-super.org/examples/process/bpmo/v2.0.1/examples#Process_ContentProvision' />
  </attributeValue>
  - <attributeValue name='http://ip-super.org/ontologies/process/bpmo/v2.0.1#hasPartnerWebService'>
    <value type='http://ip-super.org/examples/process/bpmo/v2.0.1/examples#SemanticCapability_ContentRequester_WSMO' />
  </attributeValue>
  - <attributeValue name='http://ip-super.org/ontologies/process/bpmo/v2.0.1#hasInputDescription'>
    <value type='http://ip-super.org/examples/process/bpmo/v2.0.1/examples#SemanticCapability_ContentRequestMessage' />
  </attributeValue>
  - <attributeValue name='http://ip-super.org/ontologies/process/bpmo/v2.0.1#providesCapability'>
    <value type='http://ip-super.org/examples/process/bpmo/v2.0.1/examples#SemanticCapability_ContentRequestOperation' />
  </attributeValue>
  - <attributeValue name='http://ip-super.org/ontologies/process/bpmo/v2.0.1#hasPartnerRole'>
    <value type='http://ip-super.org/examples/process/bpmo/v2.0.1/examples#contentRequester' />
  </attributeValue>
</instance>
```

**Figure 5-4** Example of BPMO instance in XML WSML

As mentioned earlier, our translator uses WSML XSD as source and target models and is concerned with WSML ontology instances. A sample of a WSML XML instance is shown in Figure 5-4 (corresponding to the text format in Listing 5-5).

Basically, a WSML instance has a name, is member of a concept (class), and contains
zero or more attribute values. An attribute value refers to the name and value of the attribute. The value can refer to either another instance (type) or a primitive content. Hence, our translator rules are mainly based on transformation from and to the following WSML XML elements: instance, attributeValue and value. These correspond to the (ECORE) types Wsml!InstanceType, Wsml!AttributeValue and Wsml!WsmlAnyValue respectively, used in our rules (see Appendix F).

Listing 5-5 Example of BPMO instances

```
instance Receive_ContentRequest memberOf bpmo#Receive
  bpmo#hasName hasValue "Receive Content Request"
  bpmo#hasPartnerWebService hasValue SemanticCapability_ContentRequester_WSMO
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasInputDescription hasValue SemanticCapability_ContentRequestMessage
  bpmo#hasPartnerRole hasValue contentRequester

instance SemanticCapability_ContentRequestMessage memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/kmi/ContentProvision/RequestContentWS#contentRequestMessage"

instance contentRequester memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Content Requester"
  bpmo#hasOrganisation hasValue kmi
```

The complete specification of the translator rules is provided in Appendix F. These rules follow on the mappings given in Table 5-2. Examples of translated instances will be given in the next chapter, as part of two use cases. In the following we explain the definition of a couple of rules in order to clarify how the translator works.

In regard to the translation from BPMO to sBPEL, an easy way to follow the implemented rules is to check the BPMO concept name following ‘bpmoNamespace’ in the source pattern (after from); and the sBPEL concept name following ‘sbpelNamespace’ in the target pattern (after to). The rules also include the definitions of Helpers, which are routines in ATL that can be used with the rules.
The translation involves a number of ontology mapping cases: sBPEL Partner, Role and Conversation concepts are derived from diverse attributes in BPMO Tasks; BPMO Tasks can derive multiple chained target concepts in sBPEL; ordered elements in BPMO (from sequences and conditional branches) generate linked-lists in sBPEL; and some target concepts in sBPEL must point back to the source concept in BPMO. Otherwise, both ontologies are aligned in the capacity of handling ontological data, SWS descriptions and semantic based mappings.

Listing 5-6 shows the translator rule for BPMO GoalTask concept. The rule named goalTasks transforms instances of GoalTask in BPMO to the sBPEL instances of ExtensionActivity, SendReceive, Conversation and OutgoingInterface, including attributes. The concept SemanticCapability in BPMO translates to SemanticVariable in sBPEL when referring to the description of messages used as input or output data. Note also the use of the correspondsTo attribute to point back to the originating BPMO instance.

Listing 5-6 Translator Rules for Goal Tasks

```xml
rule goalTasks {  
  from i: Wsml!InstanceType {  
    i.isInstanceOf(thisModule.bpmoNamespace+'GoalTask')  
  }  
  to  
    anExtensionActivity: Wsml!InstanceType {  
      name<-i.name.mapNameToSbpel(),  
      memberOf<-i.memberOf->select(a|a->trim() <>  
        thisModule.bpmoNamespace+'GoalTask')  
        ->prepend(thisModule.bpelNamespace+1ExtensionActivity'),  
      attributeValue<-Sequence{correspondsToAtt, hasActivityAtt}  
    },  
    correspondsToAtt: Wsml!AttributeValue {  
      name<-thisModule.bpelNamespace+'correspondsTo',  
      value<-Sequence{hasvalueAtt}  
    },  
    hasvalueAtt: Wsml!WsmlAnyValue {  
      type<-i.name  
    },  
    hasActivityAtt: Wsml!AttributeValue {  
      name<-thisModule.bpelNamespace+'hasActivity',  
      value<-Sequence{aValue}  
    }  
}  
```
CHAPTER 5 MEDIATION AND BUSINESS PROCESS MODELLING

aValue1: Wsml!WsmlAnyValue {
  type<-i.name.mapNameToSbpel().concat('SendReceive')
},
aSendReceive: Wsml!InstanceType {
  name<-i.name.mapNameToSbpel().concat('SendReceive'),
  memberOf<-Sequence {thisModule.spelNamespace+1SendReceive'},
  attributeValue<-Sequence{i.attributeValue,
    belongsToConversationAtt,
    (if not i.inputDescAttExists() then hasInputVarAtt else
    OclUndefined endif),
    (if not i.outputDescAttExists() then hasOutputVarAtt else
    OclUndefined endif)}
},
hasInputVarAtt: Wsml!AttributeValueType {
  name<-thisModule.spelNamespace+1hasInputVariable',
  value<-Sequence{inpVarValue}
},
inpVarValue: Wsml!WsmlAnyValue {
  type<-i.name.mapNameToSbpel().concat('GoalInputSemanticVariable')
},
anInputSemanticVariable: Wsml!InstanceType {
  name<-i.name.mapNameToSbpel().concat('GoalInputSemanticVariable'),
  memberOf<-Sequence {thisModule.spelNamespace+1SemanticVariable'},
  attributeValue<-thisModule.setWsmlAttValueType(thisModule.spelNamespace+1hasSemanticType1,
    'http://www.wsmo.org/wsml/wsml-syntax#string')
},
hasOutputVarAtt: Wsml!AttributeValueType {
  name<-thisModule.spelNamespace+1hasOutputVariable',
  value<-Sequence{outVarValue}
},
outVarValue: Wsml!WsmlAnyValue {
  type<-i.name.mapNameToSbpel().concat('GoalOutputSemanticVariable')
},
anOutputSemanticVariable: Wsml!InstanceType {
  name<-i.name.mapNameToSbpel().concat('GoalOutputSemanticVariable'),
  memberOf<-Sequence {thisModule.spelNamespace+1SemanticVariable'},
  attributeValue<-thisModule.setWsmlAttValueType(thisModule.spelNamespace+1hasSemanticType',
    'http://www.wsmo.org/wsml/wsml-syntax#string')
},
belongsToConversationAtt: Wsml!AttributeValueType {
  name<-thisModule.spelNamespace+1belongsToConversation1,
  value<-Sequence{aValue2}
},
aValue2: Wsml!WsmlAnyValue {
  type<-i.name.mapNameToSbpel().concat('Conversation')
},
aConversation: Wsml!InstanceType {
  name<-i.name.mapNameToSbpel().concat('Conversation'),
  memberOf<-Sequence {thisModule.spelNamespace+1Conversation' },
  attributeValue<-Sequence{hasNameAtt, hasdescribeslnterfaceAtt}
},
hasNameAtt: Wsml!AttributeValueType {
  name<-thisModule.spelNamespace+1hasName',
  value<-Sequence{aValue}
},
xmltype: Wsml!XMLTypeDocumentRoot {
  text<-Sequence{i.name.mapNameToSbpel().concat('Conversation').getNameWithNoNamespace()}
},
aValue: Wsml!WsmlAnyValue {
  type<-http://www.wsmo.org/wsml/wsml-syntax#string',
  mixed<-xmltype.mixed
},
hasdescribesInterfaceAtt: Wsml!AttributeValueType {
  name<-thisModule.spelNamespace+1describesInterface',
  value<-Sequence{aValue3}
},
aValue3: Wsml!WsmlAnyValue {
  type<-i.name.mapNameToSbpel().concat('OutgoingInterface')
}
Listing 5-7 shows the translator rules for Mediation Tasks and Data Mediators. A 
MediationTask is translated into Assign and DataMediator is translated into Mediate 
in sBPEL. The hasDataMediator attribute (in MediationTask) is translated into 
hasAssignOperation (in Assign) correspondingly. In addition hasMediator attribute 
(in DataMediator) translates into usesDataMediator (in Mediate). sBPEL has an extra 
concept for DataMediator, which has the hasDataMediator and hasMediationService 
attributes equivalent to BPMO.

**Listing 5-7 Translator Rules for Mediation Tasks and Data Mediators**

```java
rule mediationTasks {
    from i: Wsml!InstanceType {
        i.isInstanceOf(thisModule.bpmoNamespace+'MediationTask')
    } to
    anInstance: Wsml!InstanceType {
        name<-i.name.mapNameToSbpel(),
        memberOf<-i.memberOf->select(a|a->trim() <> 
            thisModule.bpmoNamespace+'MediationTask' ) 
            ->prepend(thisModule.bpelNamespace+'Assign ')
                ->prepend(thisModule.bpmoNamespace+'Assign ')
                attributeValue<-i.attributeValue 
        ->including(correspondsToAtt)
    }, 
    correspondsToAtt: Wsml!AttributeValueType {
        name<-thisModule.bpmoNamespace+'correspondsTo',
        value<-Sequence[hasvalueAtt]
    }, 
    hasvalueAtt: Wsml!WsmlAnyValue {
        type<-i.name
    }
}
```
rule hasDataMediatorAtt {
  from aa: Wsml!AttributeValueType
  (aa.name=thisModule.bpmoNamespace+'hasDataMediator')
to
  -- work around xml-wsml serializer
  hasWorkElemAtt: distinct Wsml!AttributeValueType foreach (med in aa.value)
  name<-thisModule.bpmoNamespace+'hasAssignOperation',
  value<-aa.value )
}

rule dataMediators {
  from i: Wsml!InstanceType {
    i.isInstanceOf(thisModule.bpmoNamespace+'DataMediator')
  }
  to
  anInstance: Wsml!InstanceType {
    anInstance: Wsml!InstanceType (
      name«-i.name.mapNameToSbpel() ,
      memberOf<-i.memberOf->select(a|a->trim() <>
        thisModule.bpmoNamespace+'DataMediator')
      ->prepend(thisModule.sbpelNamespace+'Mediate'),
      attributeValue<-Sequence {i.attributeValue,
        (if not i.inputDescAttExists() then hasInputVarAtt else
          OclUndefined endif),
        (if not i.outputDescAttExists() then hasOutputVarAtt else
          OclUndefined endif)},
      hasInputVarAtt: Wsml!AttributeValueType {
        name<-thisModule.sbpelNamespace+'hasInputVariable',
        value<-Sequence{inpVarValue},
        inpVarValue: Wsml!WsmlAnyValue {
          type<-i.name.mapNameToSbpel().concat('MediatorInputSemanticVariable')
        },
      anInputSemanticVariable: Wsml!InstanceType {
        name<-i.name.mapNameToSbpel().concat('MediatorInputSemanticVariable'),
        memberOf<-Sequence {thisModule.sbpelNamespace+'SemanticVariable'},
        attributeValue<-thisModule.setWsmlAttValueType(thisModule.sbpelNamespace+'hasSemanticType',
          'http://www.wsmo.org/wsml/wsml-syntax#string')
      },
    hasOutputVarAtt: Wsml!AttributeValueType {
      name<-thisModule.sbpelNamespace+'hasOutputVariable',
      value<-Sequence{outVarValue},
      outVarValue: Wsml!WsmlAnyValue {
        type<-i.name.mapNameToSbpel().concat('MediatorOutputSemanticVariable'),
      anOutputSemanticVariable: Wsml!InstanceType {
        name<-i.name.mapNameToSbpel().concat('MediatorOutputSemanticVariable'),
        memberOf<-Sequence {thisModule.sbpelNamespace+'SemanticVariable'},
        attributeValue<-thisModule.setWsmlAttValueType(thisModule.sbpelNamespace+'hasSemanticType',
          'http://www.wsmo.org/wsml/wsml-syntax#string')
      }},
    },
  }
}

rule hasMediatorAtt {
  from aa: Wsml!AttributeValueType
  (aa.name=thisModule.bpmoNamespace+'hasMediator')
to
  hasWorkElemAtt: Wsml!AttributeValueType {
    name<-thisModule.bpmoNamespace+'usesDataMediator',
    value<-aa.value
  }
}

udmValue: Wsml!WsmlAnyValue {
  type<-aa.value->first().type.mapNameToSbpel().concat('DataMediator')
},
aDataMed: Wsml!InstanceType
BPMO as a model which represents business processes at the business level supports constructs that do not apply to sBPEL. Similar to the mappings between BPMN and BPEL (see Ouyang et al., 2006b), the mappings from BPMO to sBPEL can only be partial. For example, BPMO allows business analysts to create arbitrary cycles, which are not supported in sBPEL. The way to restrict the translation is by using only translatable constructs such as block patterns in BPMO or doing a pre-validation through the use of axioms. One approach for the translation between BPMO and sBPEL is to count on a set of axioms which can guarantee that a valid BPMO instance contains only and at least the elements required for the translation. These axioms can be contained in a separate ontology and imported for validation of the BPMO instance. This is based on the fact (see for example (Mendling et al., 2006) that there is no generic translation from a graph-based notation such as allowed in BPMO to one such as sBPEL, which provides mostly structured activities to model a process and some restricted use of links to enable a graph-based style.

Our preliminary implementation of BPMO2SBPEL cannot deal with arbitrary loops or unsynchronized branches (as is the case for all BPMN-BPEL like translations).

Our solution is restricted to structured, pattern-based elements of BPMO. That is,
implemented version translates (generate an executable sBPEL of) BPMO diagrams (instances) which contain Tasks (GoalTask, Receive, Send, MediationTask), Events and block patterns between a StartEvent and an EndEvent or within an initial sequence. The block patterns supported are: Sequence, ParallelSplitSynchronise, MultipleChoiceMerge, DeferredChoiceMerge, ExclusiveChoiceMerge, Repeat and While. Note that BPMO processes that need not be translated (executed), have no such restrictions.

5.5 Discussion and Conclusions

The main contributions of this chapter have been BPMO and the BPMO2SBPEL translator. BPMO provides a high-level model for Business Processes workflows using the results of Semantic Web Services technologies. BPMO2SBPEL translates process models from the business level to the execution level.

In comparison to the last chapter (see also the introduction in this chapter), first this chapter presents a full-fledge workflow model (BPMO) that is an evolution of the orchestration model presented in last chapter. While BPMO fulfils the view of applications as business processes, the IRS-III's orchestration model fulfils the functionality of a Web Service. Second, in this chapter we changed our ontology language from OCML to WSML. That gave us an opportunity into experimenting with the reasoning power of each language while working with the WSMO meta-model supported by both of them. While WSML provides an easier way to express mapping rules (via axioms) and is more powerful in the capability of expressing
inferred instances within rules, OCML is more powerful as an environment to execute functions and thus to implement model components (e.g. Goals, Mediators).

We created an early version of BPMO in OCML within IRS-III so that we were able to translate a simple BPMO instance into the orchestration model of IRS-III for execution. This implementation was an experiment to show the compatibility of the two SWS based workflow models and a possible execution of BPMO. It also demonstrates the compatibility between SWS descriptions between WSML and OCML. This implementation was part of a Tutorial hands-on on SBPM, available at http://kmi.open.ac.uk/projects/super/ SUPERTutorialHandsOn.zip.

BPMO facilitates semantic interoperability by modelling interaction activities using SWS descriptions of inputs, outputs and operations. Also, by referring to SWS Goals in invocation activities we facilitate finding suitable Web Services which can fulfil these activities. The activities using ontologically defined data also take advantage of the semantic mappings provided by the BPMO Data Mediators.

BPMO diagrams can be created using practical and freely available tooling with WSMO Studio. The advantage is that the BPMO modeller of WSMO Studio automatically generates BPMO instances from a workflow diagram. It uses WSML-Flight as the representation language and provides, among others, the IRIS reasoner for performing instance validation and queries.

In Section 5.4 we have shown how we can effectively translate a business-level process model annotated with BPMO to an executable process model annotated with sBPEL, using a model-driven approach based on ATL rules. Our preliminary
implementation is restricted to structured, pattern-based elements of BPMO. We believe the translation is very efficient because it is based on Eclipse software, including the language (ATL) for writing the transformation rules. In addition, the use of ontologies makes it easier to disambiguate between block-based (structured) and graph-based elements in the process workflow.

There is substantial work discussing the translation and mismatches between BPMN and BPEL (e.g. Ouyang et al., 2006; Recker and Mendling, 2006), and more generically between block and graph oriented workflow notations (Mendling et al., 2006), which have informed the implementation of BPMO as well as the BPMO2sBPEL translator. For instance, the translator uses appropriate workflow pattern representations in BPMO to avoid workflows with acyclic loops and unsynchronised branches. One main difference from those cited papers to ours, though, is that we use ontologies and extensions to support Semantic Web Services.

In comparison, OWL-S[42] (see Chapter 2, Section 2.5.2) contains a semantic-based process workflow description (i.e. the process model), which serves the same purpose as BPMO; however this model is not very rich. As pointed out in (Aslam et al., 2006), there are a number of constructs from BPEL, such as conditions, synchronization and external (event-based) choices and handlers that cannot be expressed in OWL-S.

The Semantic Web approach presented in (Mandell and McIlraith, 2003), as discussed in the literature review (Chapter 2, Section 2.8), has similar goals to our approach

[42] http://www.w3.org/Submission/OWL-S/
using BPMO as the authors there argue that the syntactic approach provided by BPEL4WS has shortcomings that limit its ability to provide seamless interoperability. They propose the use of semantic-based technologies (OWL-S) to support automated service discovery, customization and semantic translation for BPEL4WS based processes; however, their annotations for services and data are decoupled from the control-flow language. BPMO, instead, provides semantically annotated control-flow constructs coupled with semantic descriptions of data and services. In addition, BPMO workflow includes semantic data mediation via Mediation Tasks with Data Mediators, which can refer to mapping rules and mediation services.

In (Sivashanmugam et al., 2005), as discussed in the literature review (Chapter 2, Section 2.8) an approach called Semantic Process Templates (SPT) is presented, which provides templates with semantic extensions to a BPEL-compatible process workflow specification (XML-based). A semantic template is used for every activity in the process definition, in which concepts from a given ontology are used to describe inputs, outputs and operations. SPT differs from BPMO because it is a bottom-up approach, tied to a particular execution standard (BPEL), and the control-flow constructs have no ontological representation.

From the business viewpoint, business analysts can perform semantically enabled queries directly and uniformly on the business context and activities of a business process. The queries can be extended to BPMO’s translation destinations and sources throughout the process life cycle from creation to deployment, monitoring and execution. In this way reuse across the business/IT divide is facilitated and great
scalability is achieved through increased automation supported by ontology-based reasoning. In the next Chapter we will evaluate the use of BPMO and the translator via use cases.
Chapter 6 Evaluation

In this Chapter we evaluate our contributions empirically via three use cases. These use cases employ Semantic Web Services to build business applications on the Web as well as to build business processes, resolving data and process heterogeneities. Each use case is split in four parts: i) application domain scenario; ii) mediation requirements; iii) the solution using one of our contributions from previous chapters; and iv) discussion of results. In addition, we discuss the general benefits of our approach and also evaluate a number of broader research challenges having an effect on Semantic Web Services, so that we can measure our achievements in a larger scope.

6.1 Introduction

In the first use case we evaluate our mediation framework and development approach implemented in Chapter 4 by building a SWS-based application within the e-government domain in order to validate the requirements for enterprise application integration. E-government is one of the application areas in which information systems have been implemented over heterogeneous computing platforms and data formats with no central control.

In the second use case, in the telecommunication domain, we evaluate the Business Process Modelling Ontology as well as the BPMO2SBPEL translator presented in Chapter 5. We provide our solution within an application scenario which involves business process interactions among three business partners. We address the modeling and querying of the mediation process and then its translation to the execution level.
In this context, we exploit the fact that a business process can use Semantic Web Services for representing business activities.

Our third use case is taken from the mediation scenario provided by the Semantic Web Service Challenge\(^43\), an initiative aiming at the evaluation of Semantic Web based approaches to problems such as data and process mediation. The mediation scenario consists of two implemented Web Service based systems corresponding to the processes of a (hypothetical) customer and a service provider company. The participants must extend the given syntactic service descriptions with semantics and to code a component (the process mediator) to overcome the heterogeneity between the different systems. In this use case we again apply the Process Modelling Ontology as well as the BPMO2SBPEL translator for our solution.

### 6.2 Use case in the e-government domain

In this use case we describe the implementation of the prototype we have developed within the DIP project (http://dip.semanticweb.org) in the e-government domain. As reported in (Davies, 2006), government authorities in Europe need to develop ways of delivering more usable and comprehensive electronic services to their citizens cost-effectively, if they are for instance to be able to comply with the goals of e-Europe and the related growth in national e-Government imperatives and strategies.

---

\(^43\) http://sws-challenge.org
The main requirement for applications in E-government relates to the interoperability of data and processes between services provided by governmental agencies. Service providing agencies, which are geographically dispersed, have to interact and communicate common information to one another so that a decision can be made on behalf of a user (citizen). Each participating agency has its own set of information systems developed in specific platforms accessing data resources (e.g. databases) in specific format, usually built with a certain application in mind. The ability to aggregate and re-use all the information resources relevant to a given problem and further to make this available as a basis for transparent interaction with community partner organisations and individual citizens is very restricted. Furthermore, the goals of citizens using e-government services and of government providers of services are often not conceptually aligned, contributing to misunderstanding, low take up and poor relations between citizens and their governments.

The application scenario defined for this use case was named “Change of Circumstances”, which involves two agencies coordinated by Essex County Council (ECC) in UK:

- **Community Care (Social Services) in Essex County Council** - This department typically has a coordinating role in relation to a range of services from a number of providers and special responsibility for key services such as support for elderly and disabled people (e.g. day centers, transportation). It uses the SWIFT database, which contains information related to a citizen, including case assessment for social benefits and services.
• The Housing Department of Chelmsford District Council – This department handles housing services, which can provide for example catalogue information about public household equipments that people with physical impairment can order. Their database has information about citizens using public equipment (ELMS database).

In this scenario, a case worker of the Community Care department helps a citizen to report his/her change of circumstance (e.g. address) to different agencies involved in the process. In this way, the citizen only has to inform the council once about his/her change, and the government agency automatically notifies all the agencies involved. An example might be when a mother with a specific impairment moves into her daughter’s home. The case worker opens a case for the daughter (citizen) who is eligible to receive services and benefits – health, housing, etc. Multiple service providing agencies need be informed and interact.

6.2.1 Requirements for Service-Oriented Application Integration

The main characteristic of this scenario is that it requires service oriented application integration. We create a prototype application using Semantic Web Services which have integration purpose across the various agencies involved in the scenario. The services provided through the application can be shared between agencies or used to send/get information to/from one or more agencies or even third parties.

The main mediation requirements relate to the interoperability of data between services provided by different government agencies. From the scenario above we have gathered the following mediation requirements and solutions:
• **Data Mediation** - Agencies have their own databases and hence different data formats for the same concept (e.g. Address). Different data formats can be lifted to the same concept in a domain ontology. At a semantic level, different concepts can be mapped through mediators.

• **Discovery** – Agencies achieve goals in different ways. Here we can define a Goal that can be satisfied by a number of applicable Web Services developed within different agencies. The appropriate Web Service to solve a Goal can be discovered at runtime according to the defined assumptions and preconditions of each applicable Web Service.

• **Process mediation** – A mediation process is derived by composing Goals provided by different agencies. This can be done programmatically within the Web application; or provided as the orchestration model of a specific Web Service description.

### 6.2.2 Our Solution

We apply the development approach described in Chapter 4 (Section 4.6) to the scenario above, which basically consists of building the required Semantic Web Services and then developing the prototype application which uses these Semantic Web Services.

**Applying our Proposed SWS Development Approach**

Using our development approach, we played the role of the developer for creating Goal descriptions according to user requests and Web Service descriptions for the
deployed Web Services. We then created Mediator descriptions which connected domain ontologies, Goals and Web Services.

![Figure 6-1 Screen Shot of IRS-III Browser with SWS descriptions](image)

All the semantic descriptions for this use case were created in OCML using the IRS-III Browser as shown in Figure 6-1. Via the browser we can create and publish models in the SWS library. The complete set of domain ontologies and semantic descriptions can be found in Appendix G. Later in this section we give examples of these descriptions.
At the service level, we developed a set of Web Services which performed basic operations on top of the databases of the two involved agencies. These Web services were deployed into an application server provided by a partner at SAP in Germany. The WSDL files of sample services are available in Appendix G (Section G.3).

At the legacy systems level, we recreated anonymous content (due to privacy reasons) of the existing data sources for each agency involved. These databases were also deployed at SAP. We used IRS-III’s Java Publisher that enables the invocation of Java methods. In order to invoke the Web Services deployed at SAP we created client code in Java.
At the presentation level, we created a web-based interface for the *Change of Circumstances* scenario (Figure 6-2). From the interface a case worker from Essex County Council has access to functionality such as “Query Client Details” and “Create New Assessment”. A case worker can select a suitable functionality, fill in the required fields and then submit his request, which will trigger the execution of an associated Goal. For example, the *Show Available Services/Equipment* functionality shown in the interface will trigger the invocation of the *get-catalogue-data-goal* description associated to it (Listing 6-1). This Goal description declares the Goal’s input name (*has_max_client_weight*) and type (*integer*) as well as the soap/xml schema type binding (*int*). As a result of the invocation of this goal, a list of household equipment matching the value (maximum weight) entered by the user will be shown.

Listing 6-1  Semantic description of *get-catalogue-data-goal*

```
(DEF-CLASS GET-CATALOGUE-DATA-GOAL (GOAL) ?GOAL
  (HAS-INPUT-ROLE :VALUE HAS_MAX_CLIENT_WEIGHT)
  (HAS-INPUT-SOAP-BINDING :VALUE (HAS_MAX_CLIENT_WEIGHT "int")))
```

Creating Domain Ontologies used in the IRS-III SWS descriptions

Listing 6-2 shows a couple of concepts (attributes are self-explanatory) from the domain ontologies created for the application scenario (also in Appendix G). The concept *catalogue-data* is used as output type of *get-catalogue-data-goal*. Instances of these classes are created with the values of attributes provided through the user interface. Otherwise values can originate from other service invocations.
Listing 6-2 Sample concepts from domain ontologies in the Change of Circumstances Scenario

```lisp
(def-class citizen-address ()
  (has-address-key :type integer)
  (has-postcode :type post-code-string)
  (has-premise-number : type integer)
  (has-premise-name : type string)
  (has-street : type string)
  (has-locality : type string)
  (has-town : type string))

(def-class catalogue-data ()
  (has-product-code : type string)
  (has-description : type string)
  (has-cost : type string)
  (has-max-user-weight : type integer)
  (has-charging-value : type string)
  (has-product-width : type string)
  (has-product-height : type string)
  (has-product-seat-height : type string)
  (has-product-depth : type string)
  (has-technician-fit : type boolean)
  (has-product-weight : type string)
  (has-narrative-detail : type string)
  (has-essex-ss-ot-dept : type string)
  (has-main-supplier : type string)
  (has-telephone-number : type string)
  (has-fax-number : type string)
  (has-category : type string))
```

Creating a WG-mediator for mediated SWS discovery

To illustrate mediated discovery (as presented in Chapter 4, Section 4.3.1), we discuss the get-catalogue-data-web-service Web Service description (Listing 6-3), which can solve get-catalogue-data-goal (Listing 6-1). We created the get-catalogue-data-mediator (WG-mediator description) (Listing 6-4) in order to link these two descriptions (Web Service and Goal).

Listing 6-3 Semantic description of get-catalogue-data-web-service

```lisp
(def-class get-catalogue-data-web-service (web-service) ?web-service
  (has-capability :value get-catalogue-data-web-service-capability)
  (has-interface: value get-catalogue-data-web-service-interface))

(def-class get-catalogue-data-web-service-capability (capability) ?capability
  (used-mediator :value get-catalogue-data-mediator))

(def-class get-catalogue-data-web-service-interface (interface) ?interface
  (has-choreography :value get-catalogue-data-web-service-interface-choreography)
  (has-orchestration :value get-catalogue-data-web-service-interface-orchestration))

(def-class get-catalogue-data-web-service-interface-choreography (choreography))
```

Listing 6-4 WG-mediator description

```lisp
(def-class get-catalogue-data-mediator (wg-mediator) ?mediator
  (has-catalogue-data-web-service :value get-catalogue-data-web-service)
  (has-catalogue-data-goal :value get-catalogue-data-goal))
```
Note in Listing 6-3 that get-catalogue-data-web-service declares the capability (get-catalogue-data-web-service-capability) and the interface (get-catalogue-data-web-service-interface) via the corresponding attributes. The capability declares the mediator (get-catalogue-data-mediator) via the used-mediator attribute, which is defined in Listing 6-4. The interface declares the choreography (get-catalogue-data-web-service-interface-choreography), which declares the grounding (via has-grounding attribute). The grounding is a list (in LISP syntax) of elements, which IRS-III uses to ground the Web Service description to a java class (com.sap.research.dip.wp9.catalogueEntry.client.CatalogueEntryByWeightClient) and method (getCatalogueData). In this case the java class has the client code used to invoke the deployed Web Service.

Listing 6-4 get-catalogue-data-mediator wg-mediator

The get-catalogue-data-mediator (Listing 6-4) declares get-catalogue-data-goal as the source of the mediator. In this case, nor the target Web Service (optional) neither the Mediation Service have been declared. That means there is no particular data mediation between the source Goal and the applicable target Web Service. In this particular case, we make get-catalogue-data-web-service a target for that mediator by declaring the mediator in its capability (Listing 6-3). By creating this WG-mediator description, we establish that the target is applicable to solve the Goal.
Creating the orchestration model for mediated composition

To demonstrate the use of mediated composition (presented in Chapter 4, Section 4.3.2), we discuss the change-address-web-service (Figure 6-3), which is a composed Web Service used to solve the change-address-goal (Listing 6-5). The purpose of this Web Service is to provide process mediation between the two involved agencies, in order to change the address of a citizen registered in both agencies. For this purpose we provide an orchestration model as a sequence of Goals (see the illustration in Figure 6-3). The first sub-Goal change-citizen-details-goal is used to change the address of the citizen within ECC, and the second sub-Goal redirect-equipment-goal is used to change the address of the citizen within the agency providing services related to housing equipment.

![Figure 6-3 Illustration of IRS-III orchestration and mediators](image)

The change-address-web-service description can be seen in Listing 6-6. The listing omits the input and output descriptions as they are equivalent to (inherited by) the
associated Goal. The orchestration is defined in the `change-address-interface-orchestration-problem-solving-pattern` (has-body attribute) in the Web Service’s orchestration’s interface. The `orch-seq` control-flow construct in the orchestration represents the sequential execution of the contained Goals. The `return` control-flow construct represents the value to be returned to the achieving Goal. The primitive `orch-get-goal-value` retrieves the output of a Goal.

**Listing 6-5. change-address-goal Goal Description**

```lisp
(DEF-CLASS CHANGE-ADDRESS-GOAL (GOAL) ?GOAL
  ((HAS-INPUT-ROLE
    :VALUE ADDRESS_KEY
    :VALUE POST_CODE
    :VALUE PREMISE_NUMBER
    :VALUE STREET
    :VALUE LOCALITY
    :VALUE CITIZEN_TOWN
    :VALUE CITIZEN_KEY)
   (HAS-INPUT-SOAP-BINDING
    :VALUE (ADDRESS_KEY "int")
    :VALUE (POST_CODE "string")
    :VALUE (PREMISE_NUMBER "string")
    :VALUE (PREMISE_NAME "string")
    :VALUE (STREET "string")
    :VALUE (LOCALITY "string")
    :VALUE (CITIZEN_TOWN "string")
    :VALUE (CITIZEN_KEY "int")
   )
   (HAS-OUTPUT-ROLE :VALUE CHANGE_ADDRESS_ACK)
   (HAS-OUTPUT-SOAP-BINDING
    :VALUE (CHANGE_ADDRESS_ACK "string")
   )
   (ADDRESS_KEY :TYPE INT)
   (POST_CODE :TYPE STRING)
   (PREMISE_NUMBER :TYPE STRING)
   (PREMISE_NAME :TYPE STRING)
   (STREET :TYPE STRING)
   (LOCALITY :TYPE STRING)
   (CITIZEN_TOWN :TYPE STRING)
   (CITIZEN_KEY :TYPE INT)
   (CHANGE_ADDRESS_ACK :TYPE STRING)))
```

The input values of `change-address-goal` can be used by the sub-Goals `change-citizen-details-goal` (Listing 6-7) and `redirect-equipment-goal` (Listing 6-9). However, the inputs of these Goals do not match. As a result we have to define Mediators to solve the mismatches.
Listing 6-6. Excerpt of change-address-web-service Web Service Description

```lisp
(DEF-CLASS CHANGE-ADDRESS-WEB-SERVICE (WEB-SERVICE) ?WEB-SERVICE
  ((HAS-CAPABILITY :VALUE CHANGE-ADDRESS-CAPABILITY)
   (HAS-INTERFACE :VALUE CHANGE-ADDRESS-INTERFACE)))

(DEF-CLASS CHANGE-ADDRESS-CAPABILITY (CAPABILITY) ?CAPABILITY
  ((USED-MEDIATOR :VALUE CHANGE-ADDRESS-MEDIATOR)))

(DEF-CLASS CHANGE-ADDRESS-INTERFACE (INTERFACE) ?INTERFACE
  ((HAS-CHOREOGRAPHY :VALUE CHANGE-ADDRESS-INTERFACE-CHOREOGRAPHY)
   (HAS-ORCHESTRATION :VALUE CHANGE-ADDRESS-INTERFACE-ORCHESTRATION)))

(DEF-CLASS CHANGE-ADDRESS-INTERFACE-ORCHESTRATION
  (ORCHESTRATION)
  ((HAS-PROBLEM-SOLVING-PATTERN :VALUE
      CHANGE-ADDRESS-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN)))

(DEF-CLASS CHANGE-ADDRESS-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
  (PROBLEM-SOLVING-PATTERN)
  ((HAS-BODY :VALUE
      (ORCH-SEQ
       CHANGE-CITIZEN-DETAILS-GOAL
       REDIRECT-EQUIPMENT-GOAL)
     (RETURN (ORCH-GET-GOAL-VALUE REDIRECT-EQUIPMENT-GOAL))))))
```

Creating a GG-mediator

First, we created the change-citizen-details-goal-gg-mediator Mediator (Listing 6-8), which is declared by change-citizen-details-goal (used-mediator attribute), so that we can map multiple inputs of change-address-goal (the source) into the address-details input (citizen-address type) of change-citizen-details-goal (the target). The mediator uses the mediation service change-citizen-details-goal-gg-mediator-mediation-service to transform the inputs premise_number, premise_name, street, citizen_town and post_code into one instance of citizen-address. GG-Mediators have been discussed in Chapter 4, Section 4.5.1.

Listing 6-7 change-citizen-details-goal Description

```lisp
(DEF-CLASS CHANGE-CITIZEN-DETAILS GOAL (GOAL) ?GOAL
  ((HAS-INPUT-ROLE :VALUE ADDRESS-DETAILS)
   (HAS-INPUT-SOAP-BINDING :VALUE (ADDRESS-DETAILS "sexp"))
   (HAS-OUTPUT-ROLE :VALUE CHANGE_DETAILS_ACK)
   (HAS-OUTPUT-SOAP-BINDING :VALUE (CHANGE_DETAILS_ACK "xml"))
   (ADDRESS-DETAILS :TYPE CITIZEN-ADDRESS))
```
Listing 6-8 change-citizen-details-goal-gg-mediator Description

```lisp
(DEF-CLASS CHANGE-CITIZEN-DETAILS-GOAL-GG-MEDIATOR (GG-MEDIATOR)
  ((HAS-SOURCE-COMPONENT :VALUE CHANGE-ADDRESS-GOAL)
   (HAS-TARGET-COMPONENT :VALUE CHANGE-CITIZEN-DETAILS-GOAL)
   (HAS-MEDIATION-SERVICE :VALUE CHANGE-CITIZEN-DETAILS-GOAL-GG-MEDIATOR-MEDIATION-SERVICE)))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-GOAL-GG-MEDIATOR-MEDIATION-SERVICE (GOAL)
  (HAS-INPUT-ROLE
    :VALUE POST_CODE
    :VALUE PREMISE_NUMBER
    :VALUE PREMISE_NAME
    :VALUE STREET
    :VALUE CITIZEN_TOWN
  (HAS-INPUT-SOAP-BINDING
    :VALUE (POST_CODE "string")
    :VALUE (PREMISE_NUMBER "string")
    :VALUE (PREMISE_NAME "string")
    :VALUE (STREET "string")
    :VALUE (CITIZEN_TOWN "string")
  (HAS-OUTPUT-ROLE
    :VALUE ADDRESS-DETAILS
  (HAS-OUTPUT-SOAP-BINDING
    :VALUE (ADDRESS-DETAILS "xml"))
  (POST_CODE :TYPE STRING)
  (PREMISE_NUMBER :TYPE STRING)
  (PREMISE_NAME :TYPE STRING)
  (STREET :TYPE STRING)
  (CITIZEN_TOWN :TYPE STRING)
  (ADDRESS-DETAILS :TYPE CITIZEN-ADDRESS)))

Listing 6-9 redirect-equipment-goal Description

```lisp
(DEF-CLASS REDIRECT-EQUIPMENT-GOAL (GOAL)
  (HAS-INPUT-ROLE
    :VALUE CITIZEN_KEY
    :VALUE NEW_ADDRESS
    :VALUE NEW_PHONE_NUMBER
  (HAS-INPUT-SOAP-BINDING
    :VALUE (CITIZEN_KEY "string")
    :VALUE (NEW_ADDRESS "string")
    :VALUE (NEW_PHONE_NUMBER "string")
  (HAS-OUTPUT-ROLE
    :VALUE REDIRECT-EQUIPMENT_ACK
  (HAS-OUTPUT-SOAP-BINDING
    :VALUE (REDIRECT-EQUIPMENT_ACK "xml"))
  (CITIZEN_KEY :TYPE STRING)
  (NEW_ADDRESS :TYPE STRING)
  (NEW_PHONE_NUMBER :TYPE STRING)
  (REDIRECT-EQUIPMENT_ACK :TYPE STRING)
))))

(USED-MEDIATOR :VALUE CHANGE-CITIZEN-DETAILS-GOAL-GG-MEDIATOR))

Creating an OO-mediator

205
The second Mediator provided is redirect-equipment-goal-oo-mediator (Listing 6-10), which mediates between ontology elements of change-address-goal and redirect-equipment-goal. In this example, the ontology change-of-circumstances-prototype is used both as source (has-source-component attribute) and target (has-target-component attribute) since the mapping rules refer to elements of Goal descriptions within the same ontology (see appendix G). The mapping between elements is done via the mapping rule (has-mapping-rules attribute) named redirect-address-mapping. This rule maps input (attribute) values from Goal change-address-goal into the input (attribute) value new_address of Goal redirect-equipment-goal. The input values are captured via variables (e.g. ?pn) and concatenated into one value (?na) using the OCML function string-append. OO-Mediators have been discussed in Chapter 4, Section 4.5.1.

Listing 6-10 redirect-equipment-goal-oo-mediator

```
(DEF-CLASS REDIRECT-EQUIPMENT-GOAL-OO-MEDIATOR (OO-MEDIATOR) ?MEDIATOR
 ((HAS-SOURCE COMPONENT :VALUE CHANGE-OF-CIRCUMSTANCES-PROTOTYPE)
 (HAS-TARGET COMPONENT :VALUE CHANGE-OF-CIRCUMSTANCES-PROTOTYPE)
 (HAS-MAPPING RULES :VALUE
   (((redirect-address-mapping
     (NEW_ADDRESS REDIRECT-EQUIPMENT-GOAL ?na)
   IF
   (CHANGE-ADDRESS-GOAL CHANGE-ADDRESS-GOAL)
   (PREMISE NUMBER CHANGE-ADDRESS-GOAL ?pn)
   (STREET CHANGE-ADDRESS-GOAL ?s)
   (CITIZEN TOWN CHANGE-ADDRESS-GOAL ?t)
   (POST CODE CHANGE-ADDRESS-GOAL ?pc)
   (= ?na (string-append ?pn " " ?s " " ?t " " ?pc)))
))))))
```

6.2.3 Results

In this use case we mainly applied the contributions of Chapter 3 and Chapter 4, which are related to IRS-III. We played the role of the software analyst for doing requirements analysis (with Essex County Council business stakeholders) and the role
of software developer for developing the prototype application, following the
development approach presented in Chapter 4 (Section 4.7). We used the Web
Services (including WSDL) and databases created by SAP; and created domain
ontologies and service descriptions as well as the client code used by the Java
Publisher to invoke the Web Services.

The sample OO-mediator used in the e-government scenario showed that the mapping
rules can be rather complex, therefore in the future we would suggest the use of
mapping primitives as proposed in Chapter 4 (Section 4.5.5) as part of a mapping
language which would make the mappings easier to write. That could be implemented
not only in IRS-III, but also in other frameworks.

There is a concern about the number of mediators to be created and maintained, but
we believe that this is compensated by the support that mediators give to facilitate
changes in the integrated application. That is, by providing mediation we enable
services (code) to be used without changes when integrated with other services
(orchestration) or to accomplish different goals. Also, reasoning is efficient because
the mediation is performed in the scope of services connected to a goal or in the scope
of goals connected within an orchestration. The use of mediators especially improves
the time-to-market requirement of business stakeholders, as changes in the system can
be circumscribed to small components such as Mediator or Web Service descriptions.

Our implementation of Process Mediation for this use case was embedded in the
orchestration description, which allowed us to represent the functionality of a Web
Service as a sequence of Goals. Although we demonstrated how mediated
composition can be implemented at the semantic level using a simple orchestration model, we believe that a complex orchestration involving asynchronous interactions will require a framework of its own. Hence, we discuss another solution in the next use case.

Regarding the developer, a clear benefit was to be able to define the required system functionalities in term of services and then semantically describe associated data and capabilities. From the client side, it was easy to define interface functionalities and associate them with Goal descriptions. The mediation facilitated communication (bridging) between the requester view and the provider view.

There are a couple of issues when using IRS-III that we believe will be solved as easy-to-use tools become available. For example writing logical expressions (e.g. service assumptions), mapping rules and orchestrations require expertise in using OCML. However this could be improved by having specialised tools for ontology mapping and composition, which would automatically generate the mapping and orchestration constructs to help the programmers. In addition, lifting and lowering between ontology concepts and syntactic values have been implemented just for primitive values. In this case complex outputs were represented as strings (SOAP attachments in XML). A complete solution would require a component for converting complex XML data into ontology instances.

Regarding the provided development approach, we assume a data-oriented view of the business application, where data sent by the user is used to invoke Goals and then used by the selected Web Service, through the use of mediators. The invocation of
Goals is done as part of the application code via the IRS-III API. Otherwise, Web Service providers can create composite services to fulfil a functionality via the Web Service's orchestration. Nevertheless, in the next use case we will present a process-oriented view of SWS-based applications.

### 6.3 Use Case in the Telecommunications domain

In this use case we describe the implementation of the prototype we have developed within the SUPER project ([http://www.ip-super.org](http://www.ip-super.org)) in the telecommunications domain. Our use case partner was Telefonica (Spain), one of the leading integrated telecommunications operators worldwide in the provision of communication solutions, including for instance internet services for mobile phones and networks.

According to Telefonica, competition among Telecommunication (Telco) companies is increasing because of the appearance of new companies. This is the reason why operators have to expand their coverage and to offer new and more attractive services to their final customers as well as to invest in innovation to produce new services with competitive features. In addition, in this changing business context, new technologies (broadband, mobile communications, VoIP) and new opportunities that the information society offers due to the Internet use are emerging. They allow Telco operators to offer new customer ad-hoc products, which require adaptable services to the specific needs of customers. As a result, the Telco market is probably one of the fastest growing market sectors and it is modernised with the incorporation of new technologies inside these new markets. In this new market, Telco operators tend not to provide all products and services but to redistribute complementary products supplied
by third-party companies thus increasing the number of actors inside the value chain. Therefore, B2B relationships among companies are being increased to bundle more elaborated services that satisfy new customer needs and demands. On the other hand, it is important to emphasize that businesses deal with a very dynamic environment: lots of new services have to be provided, new client demands must be satisfied, better quality demands must be fulfilled and business processes have to be changed according to technology evolution and collaboration strategies.

New and more complex products appear on the market and lead to involvement of more companies in the value chain. As a consequence, in these new business models more complicated business processes are involved. They require new systems to manage the complexity involved in end-to-end business processes. Furthermore, global operators manage lots of detailed business processes. For instance Telefonica, as a global operator, has a portfolio of more than one thousand products and a product creation rate of more than one hundred products a year. This situation makes the business processes management task not trivial and increases business processes complexity. For most Telco operators, the business processes are usually in the mind of business people who try to model the processes using the language they know. However, IT people are in charge of implementing these business processes because the tools available are too complex to be used by non technical people. And so currently, there is a large gap between business process requirements and business process implementation and it is very difficult to bridge this gap, which means that business people’s expectations are not matched. In the case of Telefonica, this situation is even worse since business and IT people belong to different companies.
Although the majority of business processes are not carried out using a single monolithic application, most of the operators do have heterogeneous ad-hoc systems and multi manufacture systems: multiple independent applications from different vendors based on different platforms need to be glued together. These applications are typically heterogeneous, autonomous, distributed, immutable (systems have their own data and process models), and designed to run independently. This schema for developing new services leads to a business poor in agility and flexibility, and to higher costs for the creation and maintenance of services.
We base our prototype in the application scenario named *Digital Content* Downloading, where we a user, for example using a mobile phone, can purchase and download digital content (e.g. movies, music, games) from the Telco provider. We use a simplified scenario as example, depicted in Figure 6-4, to explain our approach. This BPMN-based diagram models the workflow of a process with four interacting partners: the Customer, the Service Provider, the License Provider and the Content Provider. The Service Provider mediates between the Customer and the License Provider and Content Provider. The Service Provider gets the content requested by the Customer by obtaining a license from the License Provider and a URL from the Content Provider. In this workflow model, the small blocks (e.g. [Service Task] Get License) represent activities to be performed by the process.

### 6.3.1 Requirements for Business Process Modelling

Enterprises expect that the introduction of new technologies in process management will help them to achieve a state where services can be easily and quickly adaptable to customer demands. The main requirement deals not only with cost reduction but also with time-to-market reduction. That is, in a competitive market, the introduction of a new product before the competitors implies anticipation to the market. All companies aim at this competitive feature because it leads to a rapid increase in revenues.

From the context above, we derive a number of requirements that a business analyst needs to fulfil. First, business analysts need to know which services or partners belong to an existing process. Second, business analysts might need to change partners or services in existing processes. Third, new processes might be created from existing
ones or from abstract processes. Fourth, existing process workflows might need to be adjusted to work with different business partners. Finally, business processes need to be translated from the business level to the execution level.

### 6.3.2 Our Solution

In this use case we design the required business application as a business process workflow, using Semantic Web Services for fulfilling business activities. We focus on the use of the Business Process Modelling Ontology - BPMO (see Chapter 5, Section 5.3) for modelling the application scenario and provide a translation from the business modelling layer to the execution layer via the BPMO2sBPEL translator (Chapter 5, Section 5.4). We also address the querying of business process activities as well as data and process mediation.

At the time of this writing, we used the BPMO Modeller tool in WSMO Studio to draw the process diagram and generate initial instances of BPMO, version 1.4. We adapted and validated the generated instance to the version 2.0.1, which is the latest version. Appendix H contains the complete BPMO instances of the Content Provision business process (Section H.1) as well as the resulting sBPEL instance from the BPMO2sBPEL translation (Section H.2). In the following we describe in more detail the BPMO instance of the Content Provision process, including modelling, translation and querying aspects.

**Modelling the Business Process Workflow**
From the Digital Content Downloading scenario presented previously (Figure 6-4) we created the Content Provision process diagram shown in Figure 6-5, which represents the Service Provider. The Content Provision process is in fact the mediation process, which interacts with the other partner processes, and is the process to be executed. According to the process workflow the following tasks take place: a request is received \((\text{Receive})\); the input is mediated (mapped to inputs of next tasks) \((\text{MediationTask})\); two invocations \((\text{Goal Tasks})\) in parallel \((\text{ParallelSplitSynchronise})\) are performed to get the license and URL of the content; the outputs are aggregated \((\text{Mediation Task})\); and finally the result is sent \((\text{Send})\) to the customer. A number of BPMO instances corresponding to the diagram elements are presented in Listing 6-11.

![Figure 6-5 BPMO Diagram of the Content Provision Process](image)

The BPMO process instance is named Process_ContentProvision, which contains the process workflow \((\text{Sequence container})\). \texttt{GoalTask_GenerateURL} represents the \texttt{Generate URL} activity and has the information needed to be executed as a Semantic Web Service. The values of attributes \texttt{hasPartnerGoal}, \texttt{hasInputDescription}, and
hasOutputDescription inform the Goal description and ontology concepts used as input and output semantic types. These can be accessed via an URI as represented by the SemanticCapability instances. In addition, the value of attribute hasPartnerRole defines the role (content provider) and organisation (telefonica) of the providing business partner.

Listing 6-11. BPMO instances of the Content Provision Process

```xml
instance Process_ContentProvision memberOf bpmo#Process
  bpmo#hasName hasValue "contentProvision"
  bpmo#hasWorkflow hasValue Workflow_1207237301424_1670399772
  bpmo#hasWSDescription hasValue SemanticCapability_Process_ContentProvision_WSMO
  upo#belongsToOrganisation hasValue telefonica
  bpmo#hasBusinessStrategy hasValue provider#strategy45
  bpmo#hasBusinessPolicy hasValue provider#policy33
  upo#hasInvolvedRole hasValue serviceProvider

instance serviceProvider memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Service Provider"
  bpmo#hasOrganisation hasValue telefonica

instance telefonica memberOf upo#Organisation
  upo#hasName hasValue "Telefonica"

instance SemanticCapability_Process_ContentProvision_WSMO memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/wsmo/ws/Process_ContentProvision#Process_ContentProvision"

instance Workflow_1207237301424_1670399772 memberOf bpmo#Workflow
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasFirstWorkflowElement hasValue Sequence_1207237301424_469283601

instance Sequence_1207237301424_469283601 memberOf bpmo#Sequence
  bpmo#hasSize hasValue 5
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasOrderedElement hasValue {OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577}

instance ParallelSplitSynchronise_1207237301424_1641434797 memberOf bpmo#ParallelSplitSynchronise
  bpmo#hasOrderedElement hasValue {OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577, OrderedElement_1207826600209_370998577}

instance UnconditionalBranch_1207237301424_310422959 memberOf bpmo#UnconditionalBranch
  bpmo#hasElement hasValue SemanticCapability_GenerateURL_WSMO

instance GoalTask_GenerateURL memberOf bpmo#GoalTask
  bpmo#hasName hasValue "Generate URL"
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasPartnerGoal hasValue SemanticCapability_GenerateURL_WSMO
  bpmo#hasPartnerRole hasValue serviceProvider
  bpmo#hasInputDescription hasValue SemanticCapability_ContentURLReq
  bpmo#hasOutputDescription hasValue SemanticCapability_ContentURLRes

instance contentProvider memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Content Provider"
  bpmo#hasOrganisation hasValue telefonica
```

215
Modelling data-flow and data mediation

The next instance worth mentioning is *MediationTask_PrepareRequest* (Listing 6-12), which maps the values of the customer request to the input values of the Goal Tasks through a couple of *DataMediator* instances. Mediation Tasks provide dataflow and the necessary mappings between process tasks. For example, *SemanticCapability_ContentURLReq* is the *outputDescription* of *DataMediator_CopyRequestToURLReq*, which after the data mediation serves as *inputDescription* of *GoalTask_GenerateURL*. In this case the value of the *hasMediationService* attribute refers to the URI of a mediation service (via *SemanticCapability*).

**Listing 6-12 BPMO instances related to data mediation in the Content Provision Process**

```
instance MediationTask_PrepareRequest memberOf bpmo#MediationTask
  bpmo#hasName hasValue "Prepare Request"
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasDataMediator hasValue [DataMediator_CopyRequestToURLReq, DataMediator_CopyRequestToLicenseReq ]

instance DataMediator_CopyRequestToURLReq memberOf bpmo#DataMediator
  bpmo#hasMediationService hasValue SemanticCapability_CopyURLReq_MediationService
  bpmo#hasInputDescription hasValue SemanticCapability_ContentRequestMessage
  bpmo#hasOutputDescription hasValue SemanticCapability_ContentURLReq

instance SemanticCapability_CopyURLReq_MediationService memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#mediationService1"

instance DataMediator_CopyRequestToLicenseReq memberOf bpmo#DataMediator
```
Generating an Executable Process Model

In the following we discuss the translation of the `ContentProvision` process to sBPEL. Listing 6-13 shows a number of resulting sBPEL instances from the translation using our BPMO2SBPEL translator (see Chapter 5, Section 5.4). The complete sBPEL file can be found in Appendix H, Section H.2.

Listing 6-13  sBPEL instances from the Translation of the Content Provision Process
The instance names in sBPEL are generated from the ones in BPMO by appending the suffix '_sBPEL'. The instance Process_ContentProvision in BPMO has been translated to Process_ContentProvision_sBPEL, an instance of SemanticProcess. A SemanticProcess has more details than the corresponding BPMO, so the attributes hasVariable and hasConversation were generated from information from other elements such as GoalTask in BPMO. The structured Sequence concept in BPMO corresponds to a Sequence concept in sBPEL, but implemented as a linked list. The ParallelSplitSynchronise in BPMO is translated to Flow in sBPEL. A GoalTask
translation generates a chain of concepts in sBPEL, which are SemanticExtension, SendReceive, Conversation and OutgoingInterface. These correspond respectively to the translation of GoalTask_GenerateURL (BPMO) to the instances GoalTask_GenerateURL_sBPEL, GoalTask_GenerateURL_sBPELSendReceive, GoalTask_GenerateURL_sBPELConversation and GoalTask_GenerateURL_sBPELOutgoingInterface. Similar chains of concepts are created for the translation of Send and Receive. The concept SemanticCapability in BPMO corresponds to SemanticVariable in sBPEL. For example, SemanticCapability_GenerateURL_WSMO_sBPEL is a SemanticVariable referring to (hasSemanticType attribute) a WSMO Goal. One useful feature of sBPEL is that it uses the correspondsTo attribute to point back to the originating BPMO instance.

Last, but not least, a MediationTask (Listing 6-14) is translated to Assign and DataMediator is translated to Mediate as in the instances MediationTask_PrepareRequest_sBPEL and DataMediator_CopyRequestToURLReq_sBPEL.

**Listing 6-14** sBPEL instances related to data mediation from the Translation of the Content Provision Process

```plaintext
instance MediationTask_PrepareRequest_sBPEL memberOf bpel#Assign
    bpel#hasName hasValue "Prepare Request"
    bpel#hasAssignOperation hasValue {DataMediator_CopyRequestToURLReq_sBPEL,
        DataMediator_CopyRequestToLicenseReq_sBPEL}
    bpel#correspondsTo hasValue "http://ip-super.org/examples/process/bpmo/v2.0.1/examples#MediationTask_PrepareRequest"

instance DataMediator_CopyRequestToURLReq_sBPEL memberOf sbpel#Mediate
    sbpel#usesDataMediator hasValue SemanticCapability_CopyURLReq_MediationService_sBPELDataMediator
    sbpel#hasInputVariable hasValue SemanticCapability_ContentRequestMessage_sBPEL
    sbpel#hasOutputVariable hasValue SemanticCapability_ContentURLReq_sBPEL

instance DataMediator_CopyRequestToLicenseReq_sBPEL memberOf sbpel#Mediate
    sbpel#usesDataMediator hasValue SemanticCapability_CopyLicenseReq_Mediator_sBPELDataMediator
    sbpel#hasInputVariable hasValue SemanticCapability_ContentRequestMessage_sBPEL
    sbpel#hasOutputVariable hasValue SemanticCapability_ContentLicenseReq_sBPEL

instance MediationTask_AggregateResult_sBPEL memberOf bpel#Assign
```
**Querying the Business Process Space**

In the following we give example of queries that can be performed over instances of a BPMO process. For example, a business analyst might be interested in knowing the names of activities belonging to a process, or the Goal associated with a Goal Task.

**Figure 6-6** Query result: all goal tasks in the Content Provision process
For example, we performed the following queries over the ContentProvision BPMO instance, using the WSML IRIS reasoner 44:

- **All GoalTasks**:

  \[\text{?goalTask}\{\text{hasName hasValue} \text{?gtname, hasInputDescription hasValue} \text{?input, hasOutputDescription hasValue} \text{?output, hasPartnerGoal hasValue} \text{?wgoal}\} \text{memberOf GoalTask}\]

- **The WSMO Goal URI of Goal Task named "Generate URL"**:  

  \[\text{?goalTask}\{\text{hasName hasValue} \text{"Generate URL", hasPartnerGoal hasValue} \text{?sc}\} \text{memberOf GoalTask and ?sc}\{\text{hasSemanticDescription hasValue} \text{?URI}\} \text{memberOf SemanticCapability}\]

The results of the queries are shown in Figure 6-6 and Figure 6-7. The result of the first query contains two items (mapping sets) corresponding to the two goal tasks (with attribute values) in the process.

![Mapping set](image)

**Figure 6-7 Query result: URI of a goal in the Content Provision process**

The only item in the result of the second query shows the Goal description URI, which is obtained from the SemanticCapability associated with

\(\text{GoalTask \_ GenerateURL}\).

---

44 IRIS is a reasoner for WSML Flight available with WSMO Studio
6.3.3 Results

In this use case we created the process workflow of a simplified business scenario, providing the BPMO model of the mediator process (the Content Provision process) based on the processes of the involved partners, since it was the process to be executed. We used the WSMO Studio tool that enables the automatic generation of BPMO instances from the process diagram, thus, facilitating the semantic annotation of business activities within the process.

Within the Content Provision process workflow we used BPMO Tasks to provide the semantic descriptions of associated Goals, Web Services and domain data. In particular, Mediation Tasks were used to enable the data-flow between different tasks.

From the application scenario implemented above we demonstrated the use of BPMO within business process management for activities such as modelling, querying, mediation and translation to the execution level. We translated the contentProvision process to sBPEL, using the BPMO2SBPEL translator.

6.4 The SWS challenge

The SWS Challenge (http://sws-challenge.org) is an initiative organized by a group of experts in the area of Semantic Web Services, which evaluates semantic technologies used by different SWS approaches to solve specific problems within given application scenarios. The challenge has been run in workshops since July 2006. The organizers manually evaluate the solution provided by the participants according to a set of criteria, awarding certificates to the successful ones.
In particular, the SWS Challenge provides an application scenario for process mediation, according to which we can discuss the benefits of our approach. Quoting the website: "In this part of the SWS Challenge scenario we focus on interoperability problems of existing systems. The aim is to show how semantic Web technologies can help to overcome the need for manual development of mediation systems. In our initial scenario description we provide relevant information about the systems involved in two forms: using current Web service description (WSDL) and natural language text annotations. Using current state-of-the-art technologies a programmer has to interpret the information given and to code components that overcome the heterogeneity between the different systems. In the SWS Challenge participants are asked to extend the syntactic descriptions in a way that their algorithms/systems can perform the necessary translation tasks in a semi or fully automatic manner".

The application scenario on mediation (see an illustration in Appendix I, Section 1.1) is about a Purchase Order process and involves three partners: the service requester (customer), company Blue, which order products; the service provider, company Moon, which sells products; and the mediator process, which must be implemented to mediate between Blue and Moon. The goal of the mediator is to map the incoming and outgoing messages between Blue and Moon and also invoke required services from Moon so that the interactions necessary to buy a product is complete. Company Blue sends a purchase order and receives an acknowledgment via the mediator. In our solution we use BPMO for modelling the main process, providing SWS references for Task descriptions, and domain ontologies for modelling data and mappings.


6.4.1 Requirements for Heterogeneous Message Exchanges

One of the characteristics of this use case is that the problem is specified at the level of message exchanges. That is, the correct behaviour of the Mediator Process can be verified by checking that messages exchanged (sent and received) between services are correct. The mediation requirements derive from:

- The messages sent by Blue do not correspond to the messages expected by Moon.

- Blue's messages have a different data format than the ones required by Moon's.

The message format from the Blue context is specified in (RosettaNet) XML Schema. The message enables Blue (as a buyer) to issue a purchase order and to obtain from Moon (as a seller) a quick response that acknowledges which of the purchase order product line items are accepted, rejected or pending. The mediator system should:

- Adequately invoke the requisite Web Services. The success is measured by the legality of the messages exchanged. It can include error handling.

- Adapt to a new problem (changes in the messages and conditions). Success is measured by checking whether code, data or neither has to be changed.

6.4.2 Our solution

A complete solution for this use case requires building a Web Service (software component) which mediates between Blue's Web Service and Moon's Web Service.
We started by studying the messages given in XML Schema and the WSDL description of the Web Services from the Moon and Blue processes. Based on these messages, we created the Web Service associated with the Moon mediator. In appendix I (Section I.4) we present the WSDL file we created for the Web Service implementation of the Mediator Process. This WSDL file was generated automatically by our local Web Service implementation using the Spring Web Services framework (http://springframework.org/spring-ws). This Web Service implementation was done to validate and test the messages to be sent and received. The WSDL specifies the messages and operations used by the Web Service.

However, in this section we focus on the use of ontology based technology to facilitate the mediation. Our aim here is to use BPMO for modelling the mediator process, referring to SWS for the process activities; create domain ontologies for modelling data, and finally define data mappings for the Mediation Tasks. We created the Mediator Process workflow diagram (Moon Mediator Process) with WSMO Studio as shown in Figure 6-8. The modeller generates an initial set of BPMO instances corresponding to the process control-flow, to which the user can add data instances for attributes using the modeller’s property editor. The complete list of BPMO instances corresponding to this diagram is presented in Appendix I (Section I.2).

The Mediator Process workflow is quite self-explanatory. Basically, we used Receive Purchase Order and Send PO Confirmation to interact with the Blue Company, modelled as Receive and Send BPMO tasks. Map Purchase Order and Map Result are
modelled as Mediation Tasks and used to extract the values needed by the Moon Web Services and Blue Company, respectively. *Search Customer, Create Order* and *Close Order* are modelled as Goal Tasks for accessing Web Service operations provided by Moon. *Add Line Item* and *Confirm Line Item* are modelled as corresponding asynchronous Send and Receive tasks, which are called in a loop (Repeat) for every item in the received Purchase Order. *StartEvent* and *EndEvent* do as indicated.

![BPMO diagram of the Moon Mediator process](image)

**Figure 6-8** BPMO diagram of the Moon Mediator process

Listing 6-15 shows a number of BPMO instances for the Moon Mediator Process. In this use case we use the graph-based control-flow constructs of BPMO (see Chapter 5, Section 5.3), which is another modelling option, where, starting with a *StartEvent*, all executable workflow elements are linked in an explicit way using *ControlflowConnector*. Each *ControlflowConnector* points to a source
WorkflowElement and a target WorkflowElement. For example, StartEvent_1 is linked to Receive task Receive_ReceivePO via ControlflowConnector_100.

Listing 6-15. BPMO instances of the Moon Mediator Process from the SWS Challenge

```xml
instance Process_MoonMediator memberOf bpmo#Process
  bpmo#hasName hasValue "Moon Mediator Process"
  upo#hasInvolvedRole hasValue moonMediator

instance moonMediator memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Moon Mediator"
  bpmo#hasOrganisation hasValue kmi

instance kmi memberOf upo#Organisation
  upo#hasName hasValue "KMi, The Open University, UK"

instance blue memberOf upo#Organisation
  upo#hasName hasValue "Blue, SWS Challenge"

instance customer memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Blue, SWS Challenge"
  bpmo#hasOrganisation hasValue blue

instance moon memberOf upo#Organisation
  upo#hasName hasValue "Moon, SWS Challenge"

instance moonCRM memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Moon Customer Relationship Management"
  bpmo#hasOrganisation hasValue moon

instance moonOM memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Moon Order Management"
  bpmo#hasOrganisation hasValue moon

instance Workflow_1217869140038_181603276 memberOf bpmo#Workflow
  bpmo#hasHomeProcess hasValue Process_MoonMediator
  bpmo#hasFirstWorkflowElement hasValue StartEvent_1

instance StartEvent_1 memberOf bpmo#StartEvent
  bpmo#hasHomeProcess hasValue Process_MoonMediator
  bpmo#hasName hasValue "Start"

instance ControlflowConnector_100 memberOf bpmo#ControlflowConnector
  bpmo#hasHomeProcess hasValue Process_MoonMediator
  bpmo#hasSource hasValue StartEvent_1
  bpmo#hasTarget hasValue Receive_ReceivePO

instance Receive_ReceivePO memberOf bpmo#Receive
  bpmo#hasHomeProcess hasValue Process_MoonMediator
  bpmo#hasPartnerWebService hasValue SemanticCapability_ReceivePO_WSMO
  bpmo#hasPartnerRole hasValue customer
  bpmo#hasSendCounterpart hasValue Send_SendPOConf
  bpmo#hasInputDescription hasValue SemanticCapability_PurchaseOrderDesc

instance SemanticCapability_PurchaseOrderDesc memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator#PurchaseOrderRequest"

instance SemanticCapability_ReceivePO_WSMO memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/RequestPOWS#RequestPOWS"

instance ControlflowConnector_200 memberOf bpmo#ControlflowConnector
  bpmo#hasHomeProcess hasValue Process_MoonMediator
```

227
As shown in Listing 6-15 we can define the role of the process using the attribute 
upo#hasInvolvedRole. We say that the role of this process is simply of a mediator 
(moonMediator) and that the Organisation kmi is playing this role (BusinessRole). In 
the same way we define the roles of the other partners blue (customer) and moon 
(moonCRM and moonOM).

In this use case we give more details about the Receive task. In Receive_ReceivePO 
we define hasPartnerRole, hasPartnerWebService, hasSendCounterpart and 
hasInputDescription. The values of these attributes are necessary to establish an 
interaction with a partner. In particular this Receive task, which gets information from 
a partner, must send information back using a SendTask, thus the use of 
hasSendCounterpart. The information received is defined by a concept in 
hasInputDescription (via SemanticCapability), which in this case is 
http://lilianathesis.Org/usecase/swsc/datamediator#PurchaseOrderRequest. This 
concept (without the namespace) is shown in Listing 6-17. All concepts used in the 
MoonMediator process (simplified for demonstration) can be found in Appendix I, 
Section 1.3.

Listing 6-16 BPMO instances related to data mediation in the Moon Mediator Process
In this use case we give an example of how to define data mappings in WSML. For example the data mapping (hasMediator attribute) 

`SemanticCapability_MapOrderRequestToOrder` defined in DataMediator

`DataMediator_MapOrderRequestToOrder` with value `OrderFromPurchaseOrderResquest` (omitting namespace) is shown in Listing 6-17.

This mapping (axiom) transforms an instance of `PurchaseOrderRequest` (used by company Blue) into an instance of `Order` (used by company Moon).

**Listing 6-17** Concepts, data mappings and instances in the Moon Mediator Process

```plaintext
class PurchaseOrderRequest
  fromRole ofType PartnerRoleDescription
  hasPurchaseOrder ofType PurchaseOrder

class PartnerRoleDescription
  hasContact ofType ContactInformation
  hasRole ofType _string
  partnerDescription ofType PartnerDescription

class PartnerDescription
  contactInfo ofType ContactInformation
  businessInfo ofType BusinessDescription
  physicalLocation ofType PhysicalAddress

class ContactInformation
  contactName ofType _string
  emailAddress ofType _string
  telephoneNumber ofType _string

class BusinessDescription
  hasName ofType _string

class PhysicalAddress
  addressLine1 ofType _string
  cityName ofType _string
  countryCode ofType _string
  postalCode ofType _string

class Order
  authToken ofType _string
  contact ofType Contact
  shipTo ofType OrderInformation
  billTo ofType OrderInformation
```

229
concept OrderInformation
  name ofType _string
  street ofType _string
  city ofType _string
  postalCode ofType _string
  country ofType _string

concept Contact
  name ofType _string
  telephone ofType _string
  email ofType _string

relation MapOrderRequestToOrder( impliesType PurchaseOrderRequest, impliesType Order)

axiom OrderFromPurchaseOrderRequest
  definedBy
  ?request[fromRole hasValue ?pr] memberOf PurchaseOrderRequest
  and ?pr[partnerDescription hasValue ?pd] memberOf PartnerRoleDescription
  and ?pd[contactInfo hasValue ?ci, businessInfo hasValue ?bd, physicalLocation hasValue ?pl] memberOf PartnerDescription
  and ?ci[contactName hasValue ?contactName] memberOf ContactInformation
  and ?bd[hasName hasValue ?bussName] memberOf BusinessDescription
  and ?pl[addressLine1 hasValue ?adr, cityName hasValue ?c, countryCode hasValue ?co, postalCode hasValue ?pc] memberOf PhysicalAddress
  implies moonOrder(?request)[authToken hasValue "LilianaCabral", contact hasValue contact(?ci), shipTo hasValue shipTo(?bd)] memberOf Order
  and contact(?ci)[name hasValue ?contactName] memberOf Contact
  and shipTo(?bd)[name hasValue ?bussName, street hasValue ?adr, city hasValue ?c, postalCode hasValue ?pc, country hasValue ?co] memberOf OrderInformation
  and MapOrderRequestToOrder(?request, moonOrder(?request)).

instance bluePORequest memberOf PurchaseOrderRequest
  fromRole hasValue bluePartnerRole
  hasPurchaseOrder hasValue bluePurchaseOrder

instance bluePartnerRole memberOf PartnerRoleDescription
  hasContact hasValue blueContact
  hasRole hasValue "Buyer"
  partnerDescription hasValue bluePartnerDescription

instance bluePartnerDescription memberOf PartnerDescription
  contactInfo hasValue blueContact
  businessInfo hasValue blueBusiness
  physicalLocation hasValue blueAddress

instance blueAddress memberOf PhysicalAddress
  addressLine1 hasValue "North Business Center, Block 9"
  cityName hasValue "Innsbruck"
  countryCode hasValue "AT"
  postalCode hasValue "A-6020"

instance blueBusiness memberOf BusinessDescription
  hasName hasValue "Blue Company"

instance blueContact memberOf ContactInformation
  contactName hasValue "Stefan Blue"
  emailAddress hasValue "stefan.blue@blue.com"
  telephoneNumber hasValue "+43(650)89930011"
For illustration purposes, we performed the query below over the sample instances to test the OrderFromPurchaseOrderResquest axiom, using the IRIS reasoner\textsuperscript{45}:

```
"?order [authToken hasValue ?auth, contact hasValue ?c, shipTo hasValue ?s ]
memberOf Order and ?s [name hasValue ?businessName, street hasValue ?street, city hasValue ?city, postalCode hasValue ?postalCode, country hasValue ?country]
memberOf Orderlnformation"
```

This query asks about any instance of \textit{Order} with corresponding attribute values. The result of the query is presented in Figure 6-9, which basically shows a newly inferred instance of \textit{Order} (named using the function symbol \textit{moonOrder}, in accordance with the consequent of the axiom\textsuperscript{46}) to which the attribute values of instance \textit{bluePORequest} is mapped.

\textsuperscript{45} IRIS is an open source reasoner for WSML Flight, available as an integrated component of WSMO Studio.

\textsuperscript{46} This use of function symbols is safe, does not affect the decidability of WSML-Flight reasoning, is supported by the IRIS reasoner and can be reduced to a normal URI name on lowering.
We next illustrate a query that can be performed over BPMO instances related to its component workflow activities. A business analyst might be interested in knowing about tasks and partners of a specific process. For example, in the query below we ask which tasks are related to partner role customer (company Blue), with the corresponding attributes values of hasName (?name) and hasPartnerWebService (?ws):

```
"?task [bpmo#hasName hasValue ?name, bpmo#hasPartnerRole hasValue customer, bpmo#hasPartnerWebService hasValue ?ws] memberOf bpmo#Task"
```

Figure 6-10 Query Result for Finding Workflow Activities

The result of this query (Figure 6-10) contains the instances of Receive (Receive Purchase Order) and Send (Send PO Confirmation) corresponding to the interaction with company Blue as expected.

We also generated a sBPEL instance of this process using the BPMO2SBPEL translator. The main difference in this process regarding the translation is the use of graph-based elements of BPMO. The complete resulting sBPEL file is provided in
appendix I, Section 1.5. The sBPEL instance is very similar to the one generated in the previous use case, so there is no need to explain the details again.

6.4.3 Results

Although our solution for the SWS Challenge does not differ much from the solution for the previous use case, this use case presented a more complex scenario, where the requirements were given from the technical viewpoint, in the form of services and messages. Moreover, we used this use case to demonstrate some of the graph-based constructs of BPMO, and also provide some data mappings using WSML rule-type axioms. This enabled us to complete our view and discuss a number of issues on Process Mediation.

As the Mediation scenario from the SWS Challenge was presented at the technical level in terms of message exchanges; we took a bottom-up approach, starting with the data formats as given in the XML Schemas and the service descriptions in the WSDL files, and created the domain ontologies and service descriptions according to these. Details of the business was not part of the scenario description, so only brief information about the companies involved was provided with the BPMO process description.

Moreover, the Mediator process in the given scenario was created to handle completely incompatible processes in terms of behaviour (workflow) and data formats. Therefore, using BPMO to build the Mediator process facilitated design-time process mediation, making it very easy to understand the mediator and its role in the
scenario. In addition, the information provided was used to generate the executable process, by translating the BPMO instances to sBPEL.

Finally, in this use case we exemplified the use of mapping definitions in WSML used by Mediation Tasks. We also described examples of concepts used to model domain knowledge.

6.5 General Benefits of our Approaches

In the following, we discuss the general benefits of our approaches according to the results obtained from all three use cases. We refer to the requirements presented before (service-oriented application integration, business process modelling, and heterogeneous message exchanges), discussing the benefits from the knowledge modelling viewpoint and from the development viewpoint. We show that we can benefit from using the SWS models and BPMO for both data mediation and process mediation.

Regarding application integration using the IRS-III mediation framework, the main benefits are derived from the modelling of SWS data mediators either used by the Web application or as part of the Web service’s orchestration, as it allows for data mediation when invoking the desired SWS goals. In addition, IRS-III takes advantage of the SWS Data mediators during runtime in order to enable mediated discovery and mediated composition.

In general, when building distributed applications that have access to external systems or information sources, a great amount of time can be spent on understanding the
interface of these systems, the format of the information to be transferred, and how to convert from one format to another. Consequently, having a high-level view of mediation might avoid a number of unwanted effects such as hardwired solutions, application-specific solutions and changes in the existing processes. Therefore, we believe that modelling mediation aspects in an explicit way as we have done in our first use case using IRS-III can alleviate these development issues.

From the business process management viewpoint, BPMO describes a rich business process model, as demanded by the BPM community, using ontological descriptions to capture workflow and organisational concerns in a uniform and extensible manner, and reuses the results of Semantic Web Services research for the description of interaction activities.

There are various advantages for using BPMO. First, BPMO provides comprehensive semantic annotations for business processes that can be used for automated inference at the business level while facilitating the translation to the execution level. Second, BPMO provides links from the process to organisational aspects, which can be modelled independently for different domains. Third, BPMO can be used to verify at the semantic level restrictions applied to the workflow or certain process activities. Finally, BPMO facilitates the modelling of new or mediation processes based on existing ones as well as the discovery of services for goal-based activities.

BPMO facilitates semantic interoperability by modelling interaction activities using SWS descriptions of inputs, outputs and operations. These activities use ontologically
defined data for dataflow and also take advantage of the semantic mappings provided by the BPMO Data Mediators.

### 6.6 Addressing Service-Oriented Computing Research Challenges

We believe that the use cases presented in this chapter embodied the main requirements for evaluating applications requiring mediation. Nevertheless, it is useful to address here a number of broader research challenges having an effect on Semantic Web Services, so that we can measure our achievements in a larger scope. For this end, we discuss in the following subset of the research challenges (extracted in quotes below) published in a recent article by academics in the area of Service-Oriented Computing (SOC):

- **Infrastructure Support for Data and Process Integration** – According to the article, “the runtime infrastructure should provide uniform consistent access to all data by all the applications that require it, irrespective of the data format, source, or location. It should also integrate service-based applications into processes and integrate processes with other processes into end-to-end constellations that span multiple institutions”. We think mediation plays an important role in accomplishing this challenge especially because it facilitates semantic interoperability. More specifically, our specification of mediation in Chapter 4 (Figure 4-1) shows clearly how we can approach this challenge. Our first use case

---

47 Cover Feature in Computer, November 2007, published by IEEE Computer Society
on e-government in this chapter illustrates our approach. Although we have not dealt with scalability issues in this use case, we think that BPMO (Chapter 5, Section 5.3) is a good starting point on the integration of services into processes. Our use case for the Content Provision scenario in the telecommunication domain illustrates our solution, which involves applying semantic descriptions to both processes (BPMO) and services (SWS).

- Semantically Enhanced Service Discovery – According to the article, “the main challenge of service discovery is using automated means to accurately discover services with minimal user involvement. This requires explicating the semantics of the service provider and requester”. We certainly address this challenge with the support of mediation as proposed in our specification for Mediated Discovery (Chapter 4, Section 4.3.1) within business scenarios. Included in our approach are the semantic description of service requesters via Goal models and the semantic description of service providers via Web Service models in IRS-III.

- Dynamic and Adaptive Processes - According to the article, “services and processes should equip themselves with adaptive service capabilities so that they can continually morph themselves to respond to environmental changes without compromising operational and financial efficiencies...”. This issue is addressed at the heart of our process mediation approach via our Goal based orchestration or using BPMO. The use of the Goal, Web Service and Mediator models we provide in our approach give maximum flexibility to the developer or a program to modify
the combination of services in a process to suit for example new clients or providers.

- Business Driven Automated Compositions – According to the article, "SOA applications should abstract away the logic at the business level from non-business related aspects at the system level. This abstraction should enable the composition of distributed business processes... ". This challenge is addressed in our approach using BPMO, where we represent the interactions between processes at the business level. The translator (Section 4.6.2, Chapter 4) does the job of generating a representation of the process at the system level, which is further configured for execution.

- Engineering of Service Applications – According to the article, "SOA applications require a service-oriented engineering methodology that enables modeling the business environment...; translates the model into service design; deploys the service system... ". Our approach addresses partially this challenge in two ways: we provide guidelines on how to build applications out of Semantic Web Services (Section 4.7, Chapter 4); and we provide a solution into modeling and translating processes from the business level to the execution level via BPMO and the translation to SBPEL. In SUPER, a full-fledged methodology for semantic business process modelling, configuration, execution and analysis is being developed.
Chapter 7  Conclusions and Future Work

Semantic Web Services have evolved into a recognized research domain, as there is now at least three service ontologies submissions to W3C, namely OWL-S, WSMO and SAWSDL, a SOA based execution environment specification for SWS submitted to OASIS as well as the SWS Challenge initiative on evaluation of SWS approaches, which exemplify the support of standardization bodies and community building around this area. However, there have been a number of fundamental issues regarding infrastructure and deployment of this immature technology which we have addressed in this work to the advancement of the state of the art of this area.

The main contribution of this thesis has been the modelling of mediation aspects and implementation of corresponding mediation mechanisms in the context of Semantic Web Services, beyond current approaches, improving the capacity of systems to automatically locate, select, compose and invoke Web Services as well as facilitating the work of developers to build applications and processes based on SWS while aiming at the semantic interoperability of heterogeneous resources on the Web.

Within the scope of building business processes and applications using Semantic Web Service technologies for integrating data and processes across domains and organisations, mediation plays a crucial, but varied role. For example, an application system which searches for available services, should overcome all heterogeneities arising from having services implemented in diverse platforms, input data in different formats, restrictions over the user data, restrictions over process behaviour, results in different formats and so
Thus, with SWS mediation we aim at solving the inherent problem of heterogeneity of resources in global and open markets such as the Web. Ideally, service providers and service requesters should be able to interact without having to resort to changes in existing systems or complex programming efforts when market requirements or business partners change. However, as observed from the literature review (Chapter 2), mediation is a very complex task, hardly achieved automatically out of the realm of controlled formalisms. Our effort in this thesis has been in the direction of providing developers with automated design-time support to modelling components and building data mappings or mediation processes that are used at runtime, by exploring emergent semantic technologies.

In Chapter 4 we defined a semantic-enabled framework for SWS mediation around a Mediator ontology, which captures the roles of mediation for a number of activities. This framework is able to bridge at the semantic level between a user viewpoint and a provider viewpoint, perform mappings between different conceptual types, and transform values as required. This framework combines knowledge-level descriptions of mediators with mediation handling components and mapping mechanisms within a semantic broker. In particular, we extend the notion of mediation from the knowledge modelling area, in which mediation is used in conjunction with reusable components for knowledge modelling (Motta, 1999). The overall implementation is based on the conceptual definition of a comprehensive SWS framework (Chapter 3), which is part of a top-down approach to developing Semantic Web Services. The mediation mechanisms use the
semantic models of several elements so that mappings can be performed at the semantic level. The fact that mediation is not a standalone activity, that is, mediation takes places between defined entities, requires that we identify the involved elements and requirements of service requesters and service providers. Accordingly, we implemented an architecture taking these requirements into account for the type of activities that take place during SWS execution. In addition our work described an approach to building applications using Semantic Web Services (Chapter 4, Section 4.6), which demonstrates how developers can define and use mediators.

In Chapter 5 we tackled the problem of process mediation within the business process management (BPM) domain, in which Semantic Web Services can be composed into business processes. We developed the Business Process Modelling Ontology (Chapter 5), taking into consideration the translation of processes from the business level to the execution level. BPMO enables the semantic annotation of business process workflow models based on Semantic Web Services, facilitating the modelling of mediation processes based on existing ones as well as the modelling of mediation tasks. Moreover, BPMO goes beyond the mediation realm into supporting various BPM activities, from modelling and querying to execution (via translation) as well as the discovery of services for goal-based activities. BPMO based process activities (Tasks) can refer to semantically annotated data, goals and services as well as incorporate heterogeneous data through semantic mappings.
Both, the Mediator ontology (described in Chapter 4) and BPMO (described in Chapter 5) are the pillar contributions of this thesis, fulfilling the need for explicitly representing mediation aspects that can be shared and reused by developers as well as used at runtime via semantic-enabled engines. Furthermore, these ontologies play a role in defining how Semantic Web Service technologies can be used in activities such as service discovery, service composition, service invocation, process modelling and process execution, in the presence of heterogeneities.

With our ontology-based approach to mediation we account for the requirements of business organisations who seek more open and transparent techniques for interoperating with heterogeneous partners as well as for bridging the gap between the business view and the IT view when implementing software solutions. In Chapter 6, we evaluated our contributions via use cases using implemented systems, as well as ontologies and instances created both in OCML and WSML, and respective reasoners. Two of the use cases accounted for the business requirements of real companies (in the government and in the telecommunication sectors) and the third one was based on a running mediation scenario from a SWS evaluation initiative.

The legacy of this work lies in bringing mediation of data and processes over the Web beyond the syntactic level to the semantic level within an integrated approach, solving the problems of semantic heterogeneity in one level and semantic interoperability in a broader scope. Looking outwards, I believe this research shall contribute to the vision of
the next generation Web: to transform the e-business landscape into an interoperable community supporting dynamic integration of machine understandable services.

In the following sections, we provide our final conclusions in view of our research questions (presented in Chapter 1) and contributions. A more thorough analysis of the technologies produced and discussion of results was given per Chapter.

### 7.1 Conclusions according to the Research Questions

**How can mediation at the semantic level improve the interoperability of Web Services?**

We have described Web Services in detail (in Chapter 2, Section 2.2) and concluded that the interoperability provided by current Web Service technologies and standards for solving business application integration is very limited. The main problem is that syntactic descriptions are not sufficient for the automation of activities such as discovery, composition and execution of Web Services. Although SWS approaches (as described in Chapter 2, Section 2.5) tackle this problem, mediation is needed to align and communicate different conceptualizations between requesters and providers of services. In order to enable semantic interoperability, Web Services must have their capabilities described and aligned to the capabilities of other services as well as to the requests of software clients within integrated application scenarios. In this context, we have shown how the ontologies we have produced can provide a way to describe service capabilities, processes and alignments through mediation.
What is the role of mediation within Semantic Web Services?

Broadly speaking, mediation can be viewed as a brokering activity for software application systems. However, as we discussed in the literature review (Chapter 2), the role of mediation has to be considered within the context where mediation takes place, which can for instance vary from information systems, to intelligent agents or process management. Within the SWS context we considered not only data but also the entities involved in the mediation, such as services, goals and processes which were all semantically described. As part of a Semantic Web Service infrastructure (Chapter 4) we have broken down mediation in many levels and mechanisms, considering the interactions between service requesters and service providers. In Chapter 5 we considered the context of business process workflows and the interaction between partner processes using SWS. As will be discussed below (Section 7.2), we considered the role of mediators as bridges within components for knowledge modelling; the role of mediators as handling components for a number of brokering activities; and the role of mediators as mapping mechanisms for service’s input and output alignment and transformation.

What activities in Semantic Web Services need mediation?

Mediation takes place during a number of runtime activities such as discovery, composition and invocation of Semantic Web Services. Essentially, every matchmaking (e.g. via discovery) or interaction activity (e.g. via operation calls or workflow execution) between Web Services in which there are mismatches, must be mediated. In this view,
we specified in Chapter 4 (Section 4.3) the operational semantics of a number of mediators for a generic SWS application scenario and their use especially for discovery and composition. In Chapter 5, we modelled mediation tasks as part of process workflow activities that can be translated for execution. As in our approach all these activities are supported by the knowledge models we have developed, we devoted Chapter 6 to the application of these technologies across use cases from the application development viewpoint.

**What types of mismatches can be handled by mediation components?**

As Semantic Web Services provide the semantic descriptions of client goals, Web Service capabilities as well as domain ontologies, data mismatches can be handled between any of these components. In IRS-III (Chapter 4, Section 4.5) we distinguish between types of mediators according to the source and target components, in particular, we implement OO-Mediators, WG-Mediators, and GG-Mediators which are used according to SWS activities (data mediation, discovery and composition). Thus, by having ontological meta-models representing different types of mediators, we can provide specialised meditation handling components. Our implementation in chapter 4 shows that mismatches can be solved by providing mapping rules or mediation services from a source component to a target component. Similarly, in Chapter 5, we use these mechanisms within process activities in order to support process mediation.
7.2 Conclusions according to Contributions

7.2.1 Mediators as components for Knowledge Modelling

In Chapter 4 we used ontologies to explicitly define mediators for Semantic Web Services. This has been one main difference between the OWL-S approach and the WSMO approach, as described in Chapter 2. Our approach with IRS-III in Chapter 4 is the only one which implements the WSMO mediator meta-model as a first class knowledge model integrated with the ones for Goal and Web Services. By using the mediator models we could define semantic bridges between the SWS elements, and as a result exploit reasoning capabilities for dealing with heterogeneities and integration of business services and processes.

7.2.2 Mediators as Software Components

A common role of a mediator is that of a software component which translates between the protocols of incompatible peers through specific mediating tasks. In IRS-III (Chapter 4), we used this definition when implementing mediation handlers which are able to reason over the mediation descriptions for different activities such as selection and composition. In a similar fashion, in Chapter 5 we assumed that a semantics-enabled workflow engine would be able to execute the workflow model representing the orchestration of several SWS along with the mappings provided via mediation tasks in order to handle the mismatches between interacting partner processes.
7.2.3 Mediators as Mapping Mechanisms

Yet another role of mediators within the SWS context is that of providing mappings or transforming instances between domain and service ontologies. In our work we particularly explored the capabilities of OCML and WSML (Chapter 3, Section 3.4) to provide mapping rules that could be used as part of the IRS-III Mediator ontology and BPMO respectively. We focused on the provision of correct mappings at design time as this is a requirement for business to business integration. In Chapter 4 we described our approach within IRS-III for providing mapping rules as well as mediation services. In Chapter 6, we provided examples of mapping rules both in OCML and WSML for the use cases. Providing specific algorithms which can extract alignments from ontologies was not the focus of our approach, but such algorithms could be integrated into a design-time tool for automated support during application development.

7.2.4 Mediation as brokering

The broadest role of a mediator is that of a broker. In this sense, we can view IRS-III (Chapter 3) as the SWS broker or mediator performing various tasks at runtime, but also supporting a range of other design time activities, such as publishing and querying, as has been demonstrated in the use cases in Chapter 6. Similarly, in the context of Business Process Management (Chapter 5), the runtime broker is the process engine. However, a number of essential activities is performed before (and after) the process runs. In this thesis, we focused on the process modelling activity using BPMO (see Chapter 5,
Section 5.3) and the translation of BPMO to an ontology of executable processes (Chapter 5, Section 5.4).

7.2.5 Application Development

A great part of our work focused on application development using the technologies we produced. In particular we outlined an approach to building applications using SWS when using IRS-III in Chapter 4 (Section 4.6) and based our evaluation (Chapter 6) on application development activities. Accordingly, our main users are developers or system analysts with some expertise in knowledge modelling.

7.3 Final Considerations

When considering mediation approaches for the integration of heterogeneous components or data resources, we can find a variety of techniques from different research communities ranging from techniques that map heterogeneous resources to a unified ontology to techniques using mediation components for handling transformations between different message protocols. In this thesis we have focused on the ontological modelling of SWS components and integration aspects of mediation rather than on mapping algorithms. As a result, we have investigated how a mediation framework can handle semantic descriptions for solving mismatches during selection, composition and invocation of services. In particular, we advocate a top-down approach where rich semantic descriptions can refer to any syntactically described Web Service instead of a bottom-up approach where semantics are tied to a specific standard or notation.
CHAPTER 7 CONCLUSIONS AND FUTURE WORK

The strength of our approach derives partially from the strength of the reasoning system behind the framework. In particular, knowledge level systems gain from the ability to reuse components and the ability to express and reason over complex relations between entities. With our use cases, we have shown a number of ways in which reasoning is applied to mediation tasks in the context of Semantic Web Services for solving real problems, with real Web Services, based on real business scenarios.

7.4 Future Work

Because of the need for designing and consolidating existing ongoing work and unsolved issues concerning the infrastructure for mediation, many questions outside the scope of this work still remain unanswered such as the robustness of our semantic approach against syntactic approaches and comparison with approaches that handle mediation in a different manner. We also could not check the execution of translated BPMO processes since there was no appropriate execution engine available. In addition, maintenance issues can also be studied, such as the ability to keep a library of mediators always consistent and accurate as Web Services come and go. Finally, there are a number of tools that can be developed to support developers in generating correct mappings and service descriptions.

Overall, the legacy of this work is encouraging in terms of the new path it paves for further research on mediation of Semantic Web Service.
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards. International consortium that drives the development, convergence, and adoption of e-business standards. Website at</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language. A XML-based standard language for ontologies submitted to W3C. Built on top of RDF-S and based on the Description Logic formalism. Website at <a href="http://www.w3.org/TR/OWL">http://www.w3.org/TR/OWL</a></td>
</tr>
<tr>
<td>OWL-S</td>
<td>Web Ontology Language for Services. An ontology for service description based on OWL. It is a W3C submission. Website at <a href="http://www.w3.org/2002/ws/OWL-S">http://www.w3.org/2002/ws/OWL-S</a></td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework. A W3C standard for data markup. Website at <a href="http://www.w3.org/TR/rdf-concepts">http://www.w3.org/TR/rdf-concepts</a></td>
</tr>
<tr>
<td>RDF-Schema</td>
<td>RDF Vocabulary Description Language. Website at <a href="http://www.w3.org/TR/rdf-schema">http://www.w3.org/TR/rdf-schema</a></td>
</tr>
<tr>
<td>Semantic Description</td>
<td>Semantic markup or annotation of services (in particular) using an ontology.</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration. An OASIS Standard which specifies protocols for creating a registry for Web services, methods for controlling access to the registry, and a mechanism for distributing or delegating records to other registries. Website at <a href="http://www.uddi.org">http://www.uddi.org</a>.</td>
</tr>
<tr>
<td>UPML</td>
<td>Unified Problem Solving Method Language. Framework created as part of IBROW.</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language. W3C standard format for data interoperability.</td>
</tr>
<tr>
<td>W3C</td>
<td>WWW Consortium. Web standardization body. Website at <a href="http://www.w3c.org">http://www.w3c.org</a></td>
</tr>
<tr>
<td>Web Service</td>
<td>According to W3C, a software component which uses XML based standards for communication over the Web. It is associated with an</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>WWW</td>
<td>The World Wide Web. Also referred as to the Web.</td>
</tr>
</tbody>
</table>
Bibliography


Language Reference, W3C Proposed Recommendation.
http://www.w3.org/TR/owl-ref/.


Web Conference (ESWC 2006), Budva, Montenegro. LNCS 4011, Springer

Reasoning: the Right Level of Abstraction for Knowledge Acquisition. Int.
Journal Man-Machine Studies, 26(2), pp. 231-243

In Workshop on Mediation in Semantic Web Services (MEDIATE 2005) in
conjunction with the 3rd International Conference on Service Oriented Computing
(ICSOC 2005), Amsterdam

Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Tanasescu, V., Pedrinaci, C. and

256


BIBLIOGRAPHY


262


Appendix A  The IRS-III Service Ontology

This appendix contains the models for Goal, Web-Service and Mediator in the IRS-III Service Ontology.

(def-class meta-invokable-entity () ?x
  :iff-def (or (?x invokable-entity)
  (subclass-of ?x invokable-entity)))

(def-class invokable-entity ()
  "Captures the input and output roles which are used within goals and Web services."
  ((has-input-role :type role)
  (has-output-role :type role)))

(def-class meta-wsmo-entity () ?x
  :iff-def (or (?x wsmo-entity)
  (subclass-of ?x wsmo-entity)))

(def-class wsmo-entity ()
  ((has-non-functional-properties :type non-functional-properties)))

(def-class effect (unary-kappa-expression))

(def-class pre-condition (unary-kappa-expression))

(def-class post-condition (unary-kappa-expression))

(def-class assumption (unary-kappa-expression))

(def-class meta-goal () ?x
  :iff-def (or (?x goal)
  (subclass-of ?x goal)))

(def-class goal (wsmo-entity invokable-entity)
  ((has-mediator :type meta-mediator)
  (has-post-condition :type post-condition)
  (has-effect :type effect)))

(def-class meta-capability () ?x
  :iff-def (or (?x capability)
  (subclass-of ?x capability)))

(def-class capability (wsmo-entity)
  ((used-mediator :type meta-oo-mediator)
  (has-pre-condition :type pre-condition)
  (has-post-condition :type post-condition)
  (has-effect :type assumption)
  (has-effect :type effect)))

(def-class meta-web-service () ?x
  :iff-def (or (?x web-service)
  (subclass-of ?x web-service)))

(def-class web-service (invokable-entity wsmo-entity)
  ((has-capability :type meta-capability)
  (has-interface :type meta-interface)
  (used-mediator :type meta-oo-mediator)))

(def-class meta-interface () ?x
  :iff-def (or (?x interface)
  (subclass-of ?x interface)))

(def-class interface (wsmo-entity)
  ((has-choreography :type meta-choreography)
  (has-orchestration :type meta-orchestration))

269
(used-mediator :type meta-oo-mediator)))
(def-class meta-mediator (wsmo-entity))
(def-class meta-oo-mediator (meta-mediator) ?x
  :iff-def (or (= ?x mediator)
              (subclass-of ?x meta-mediator)))
(def-class mediator (wsmo-entity))
(def-class mapping-rules (list))
(def-class oo-mediator (mediator)
  (has-mapping-rules :type mapping-rules)
  (has-source-component :type meta-ontology)
  (has-target-component :type meta-ontology))
(def-class meta-wg-mediator (meta-mediator) ?x
  :iff-def (or (= ?x wg-mediator)
              (subclass-of ?x wg-mediator)))
(def-class wg-mediator (mediator)
  (has-source-component :type meta-goal-or-meta-web-service)
  (has-target-component :type meta-goal-or-meta-web-service)
  (has-mediation-service :type meta-goal))
(def-class meta-ww-mediator (meta-mediator) ?x
  :iff-def (or (= ?x ww-mediator)
              (subclass-of ?x ww-mediator)))
(def-class ww-mediator (mediator)
  (has-source-component :type meta-web-service)
  (has-target-component :type meta-web-service)
  (has-mediation-service :type meta-goal))
(def-class meta-gg-mediator (meta-mediator) ?x
  :iff-def (or (= ?x gg-mediator)
              (subclass-of ?x gg-mediator)))
(def-class gg-mediator (mediator)
  (has-source-component :type meta-goal)
  (has-target-component :type meta-goal)
  (has-mediation-service :type meta-goal))
(def-class goal-or-web-service ?x
  :iff-def (or (goal ?x)
             (web-service ?x)))
(def-class meta-goal-or-meta-web-service ?x
  :iff-def (or (= ?x goal-or-web-service)
             (number-of ?x goal-or-web-service)))
Appendix B  IRS-III Internal Components for SWS

Data Mediation

This appendix contains implementation code of the IRS-III internal components related to Data Mediation.

(defun ip::internal-solve-goal (ontology goal-type role-value-pairs
  &optional (call-strategy :first))
  ;; (setf oo ontology gg goal-type rr role-value-pairs)
  (ocml::with-ocml-thread-safety ()
    (let ([result ontology-nil ontology]
      (ocml::with-thread-safety-open ()
        (ocml::select-ontology ontology)
        (let ((result ontology-for-invocation
          (ocml::find-all-web-services-with-mediators-which-solve-goal goal-type
            ontology)))
          ;; (format t "contenders ~a~%" web-service-contenders-with-mediators)
          (let ((applicable-web-services
            (mapcan #\'(lambda (web-service-and-mediator)
              (destructuring-bind (web-service mediator)
                web-service-and-mediator
              (setf ontology-for-invocation
                ontology-for-invocation ontology web-service mediator))
            web-service-contenders-with-mediators)))
            (let ((result
              (suitable-web-service? ontology-for-invocation
                goal-type role-value-pairs
                web-service)))
              (when result
                (list result)))
            (unless applicable-web-services
              (error "No applicable web services."))
            (setf result
              (case call-strategy
                ((:first) (invoke-service (third (car applicable-web-services))
                  (second (car applicable-web-services))))
                ((:all) (mapc #\'(lambda (applicable-web-service)
                  (invoke-service (third applicable-web-service)
                    (second applicable-web-service))
                applicable-web-services))))
            (when (boundp '*achieve-goal-results*)
              (push (list (list ontology goal-type) result) *achieve-goal-results*)))
            (values result ontology-for-invocation)))
            (case call-strategy
              ((:first) (invoke-service (third (car applicable-web-services))
                (second (car applicable-web-services))))
              ((:all) (mapc #\'(lambda (applicable-web-service)
                (invoke-service (third applicable-web-service)
                  (second applicable-web-service))
              applicable-web-services))))
            (when (boundp '*achieve-goal-results*)
              (push (list (list ontology goal-type) result) *achieve-goal-results*)))
            (values result ontology-for-invocation))))
            (case call-strategy
              ((:first) (invoke-service (third (car applicable-web-services))
                (second (car applicable-web-services))))
              ((:all) (mapc #\'(lambda (applicable-web-service)
                (invoke-service (third applicable-web-service)
                  (second applicable-web-service))
              applicable-web-services))))
            (when (boundp '*achieve-goal-results*)
              (push (list (list ontology goal-type) result) *achieve-goal-results*)))
            (values result ontology-for-invocation))))
            (ocml::suitable-web-service-goal ocml::find-web-services-for-goal ocml::mediate-input-role-values-goal)
            (defun internal-irs-broker-goal-p (x)
              (find x *internal-irs-broker-goals*))
            (defun mediate-role-pairs (ontology goal-type web-service actual-role-pairs)
              (let* ([role-names (mapcar #\'car actual-role-pairs))
                [role-values (mapcar #\'second actual-role-pairs))
                (result (ip:internal-solve-goal
                  ontology 'ocml::mediate-input-role-values-goal
                  (list (ocml:has-goal goal-type)
                    (ocml:has-mediator-ontology ontology)
                    (ocml:has-web-service web-service)
                    (ocml:has-input-role-values role-values))]
                (role-values (car result)))
                (combined-oo-mediators-ontology (second result)))
                (values

271
(mapcar #'(lambda (role-name role-value)
            (list role-name role-value))
       role-names role-values
       combined-oo-mediators-ontology)))

(def-class mediate-input-role-values-goal (goal)
"This goal transforms the inputs of a goal to the inputs of a web service."
  ((has-input-soap-binding
    :value (has-goal "sexpr")
    :value (has-web-service "sexpr")
    :value (has- mediation-ontology "sexpr")
    :value (has-input-role-values "sexpr")
    :value (has-output-soap-binding
    :value (has-mediated-input-role-values "sexpr"))
  (has-input-role :value has-goal
    :value has-web-service
    :value has-input-role-values
    :value has- mediation-ontology)
  (has-output-role
    :value has-mediated-input-role-values)
  (has-goal :type goal-type)
  (has-web-service :type web-service-type)
  (has-input-role-values :type list)
  (has- mediation-ontology :type ontology)
  (has-mediated-input-role-values :type list)
  (has-post-condition
    :value (kappa (goal)
                      (are-mediationally-equivalent
                        (has-role-value goal has-input-role-values)
                        (has-role-value goal has-mediated-input-role-values))))

(def-class mediate-input-role-values-mediator (wg-mediator)
  ((has-source-component :value mediate-input-role-values-goal)))

(def-class mediate-input-role-values-web-service (web-service)
  ((has-internal-method :value irs-input-role-values-mediator)
   (has-capability :value mediate-input-role-values-capability)))

(def-class mediate-input-role-values-capability (capability)
  ((used-mediator :value mediate-input-role-values-mediator)))

(defun get-mapping-rule-spec-relation (mapping-rule-spec)
  (caar (second mapping-rule-spec)))

(defun create-oo-mediator-ontology (source-ontology target-ontology)
  (let ((new-ontology-name (intern (format nil "~a-TO-~a-MEDIATOR-ONTOLOGY" source-
                                      target-ontology))
         (find-package "OCML")))
    (eval `(ocml::def-ontology-internal ',new-ontology-name
               ,(format nil "Ontology to mediate from
               ~a TO ~a" source-ontology target-ontology)
               ',(includes ,source-ontology ,target-
               ontology)
               :type :web-service
               :author "john")))

(defun create-combined-oo-mediator-ontologies (web-service ontologies)
  (let ((new-ontology-name (intern (format nil "~a-COMBINED-MEDIATOR-ONTOLOGY" web-
                                        service))
         (find-package "OCML")))
    (eval `(ocml::def-ontology-internal ',new-ontology-name
               ,(format nil "Combined Ontology for ~a" web-
               service)
               ',(includes ,ontologies
               :type :web-service
               :author "john")))
(defun create-mapping-rules (mapping-rule-specs)
  (mapc #'(lambda (mapping-rule-spec)
    (format t "def-rule ~a ~a~%" (car mapping-rule-spec) (cdr mapping-rule-spec)))
    mapping-rule-specs))

(defun run-mapping-rules (mapping-rule-specs input-role-values)
  (mapc #'(lambda (mapping-rule-spec)
    (setf input-role-values
      (mapcar #'(lambda (input-role-value)
        (format t "run ~a~%" 
          (get-mapping-rule-spec-relation mapping-rule-spec) input-role-value ?x))
        input-role-values))
    mapping-rule-specs)
  input-role-values)

(defun run-oo-mediator (oo-mediator input-role-values)
  (let ((source
    (web-onto: :findany '?x 
      (ocml::wsmo-mediator-source-component ,oo-mediator ?x)))
  (target
    (web-onto: :findany '?x 
      (ocml::wsmo-mediator-target-component ,oo-mediator ?x)))
  all-mapping-rules
  (mapping-rules
    (web-onto: :findany '?x 
      (ocml::wsmo-mediator-mapping-rules ,oo-mediator ?x))))
  (when (and source target mapping-rules)
    (let ((new-ontology (create-oo-mediator-ontology source target)))
      (ocml::select-ontology new-ontology)
      (setf all-mapping-rules (apply #'append mapping-rules))
      (create-mapping-rules all-mapping-rules)
      (values new-ontology
        (run-mapping-rules (car mapping-rule-specs) input-role-values))))

(defun ocml::irs-input-role-values-mediator (ontology web-service)
  ;;(setf xoo ontology xxw web-service)
  (ocml::with-ocml-thread-safety ()
    (ocml::select-ontology ontology)
    (let ((instance (ocml::find-current-instance web-service)))
      (when instance
        (ocml::select-ontology (ocml::name (ocml::home-ontology instance)))
        (let* ((goal-instance
          (web-onto: :findany '?x 
            (ocml::suitable-web-service ?x ,web-service))))
          (goal-to-mediate
            (web-onto: :findany '?x 
              (ocml::has-goal ,goal-instance ?x)))
            (ontology-for-mediation
              (web-onto: :findany '?x 
                (ocml::has-mediation-ontology ,goal-instance ?x)))
                (web-service-to-mediate
                  (web-onto: :findany '?x 
                    (ocml::has-web-service ,goal-instance ?x)))
                mediators wg-mediators oo-mediators ;;used-mediator mediation-service)
new-oo-mediator-ontologies nil)
new-oo-mediator-ontology
(input-role-values
(web-onto::findany
'(?x
(oocl::has-input-role-values ,goal-instance ?x)))))
(oocl::select-ontology ontology-for-mediation)
(setf mediators
(web-onto::findany
'(?x
'(oocl::wsmo-web-service-all-used-mediators-for-goal
,web-service-to-mediate ,goal-to-mediate ?x))
wg-mediators
(mapcan #'(lambda (x) (when (wg-mediator-p x)
(list x)))
mediators)
ww-mediators
(mapcan #'(lambda (x) (when (oo-mediator-p x)
(list x)))
mediators)
;;(setf gg goal-to-mediate mm mediators)
(mapc #'(lambda (oo-mediator)
(multiple-value-setq (new-oo-mediator-ontology input-role-values)
(cons (run-oo-mediator oo-mediator input-role-values)
new-oo-mediator-ontologies)))
ww-mediators)
(mapc #'(lambda (wg-mediator)
(setf input-role-values
(run-wg-mediator ontology-for-mediation ;;ontology
goal-to-mediate web-service instance
wg-mediator input-role-values)))
ww-mediators)
(list input-role-values
(if new-oo-mediator-ontologies
(create-combined-oo-mediator-ontologies
web-service-to-mediate
(remove-duplicates
(cons ontology-for-mediation new-oo-mediator-ontologies))
ontology-for-mediation)))))

(defun run-wg-mediator (ontology goal-to-mediate web-service web-service-instance
wg-mediator input-role-values)
(let ((mediation-service
(web-onto::findany
'(?x
'(and (oocl::wsmo-mediator-mediation-service
,wg-mediator ?x)))))
(if mediation-service
(let* ((web-service-input-roles
'(and (oocl::get-domain-class web-service)
(local-input-roles web-service)))
(input-roles
(input-roles goal-to-mediate))
(mediator-output-role
(output-role mediation-service))
(mediator-input-roles
(input-roles mediation-service))
(input-roles-to-mediate
(intersection input-roles mediator-input-roles))
mediated-values)
;;(format t "mediator ~s" goal-to-mediate)
(setf mediated-values
[mapcar #'(lambda (input-role-to-mediate)
(list input-role-to-mediate
(elt input-role-values
(position input-role-to-mediate input-roles))))])
274
(when (find mediator-output-role web-service-input-roles)
  (ocml::telll (,mediator-output-role ,web-service-instance ,mediated-values)))

(when (= (length input-roles-to-mediate) 1)
  ;; if only one input role is mediated then
  ;; only a single value is returned
  ;; otherwise a list
  (setf mediated-values (list mediated-values)))

(mapcar 
  #'(lambda (input-role)
      (if (find input-role input-roles-to-mediate)
          (elt mediated-values (position input-role input-roles-to-mediate))
          (elt input-role-values (position input-role input-roles)))
    input-roles))
Appendix C  IRS-III Internal components for SWS

Discovery

This appendix contains implementation code of the IRS-III internal components related to the discovery of Web Services according to a Goal.

(def-class internal-goal (goal))
(def-class internal-mediator (mediator))
(def-class internal-capability (capability))
(def-class internal-web-service (web-service)
  ((has-internal-method :type symbol)))

(def-class suitable-web-service-goal (internal-goal)
  ((has-goal :type has-goal)
   (has-actual-role-pairs :type has-actual-role-pairs)
   (has-web-service :type has-web-service)
   (has-combined-oo-mediator-ontology :type has-combined-oo-mediator-ontology)
   (has-input-soap-binding :value has-goal "sexpr")
   (has-actual-role-pairs "sexpr")
   (has-web-service "sexpr")
   (has-combined-oo-mediator-ontology "sexpr")
   (has-goal-and-web-service-instances)
   (has-goal-and-web-service-instances :type list)
   (has-output-soap-binding :value has-goal-and-web-service-instances "sexpr")
   (has-goal :type goal-type)
   (has-actual-role-pairs :type list)
   (has-web-service :type meta-web-service)
   (has-combined-oo-mediator-ontology :type ontology)
   (has-post-condition :value kappa (?goal)
     (is-suitable-for-goal
      (instantiate (has-role-value ?goal has-goal)
        (has-role-value ?goal has-actual-role-pairs)
        (has-role-value ?goal has-web-service))))
)

(def-class suitable-web-service-mediator (mediator)
  ((has-source-component :value suitable-web-service-goal)))

(def-class suitable-web-service-web-service (web-service)
  ((has-internal-method :value suitable-web-service-internal-method)
   (has-capability :value suitable-web-service-capability)))

(def-class suitable-web-service-capability (capability)
  ((used-mediator :value suitable-web-service-mediator)))

(def-class find-web-services-for-goal (internal-goal)
  ((has-input-url :value has-goal :value has-ontology)
   (has-input-soap-binding :value has-goal "sexpr")
   (has-ontology "sexpr")
   (has-internal-method :value has-internal-method)
   (has-capability :value has-capability)
   (has-output-soap-binding :value has-output-soap-binding)
   (has-association :value has-association)
   (has-association :type list)
   (has-post-condition :value has-post-condition)
   (has-goal :type meta-goal)
   (has-ontology :type ontology)
   (has-post-condition :value has-post-condition))
(kappa (?goal)
  (and (member ?web-service
       (has-role-value ?goal associated-web-services))
       (can-solve-goal
        (has-role-value ?goal has-goal) ?web-service)))

(def-class find-web-services-for-goal-mediator (wg-mediator)
  ((has-source-component :value find-web-services-for-goal!)))

(def-class find-web-services-for-goal-web-service (internal-web-service)
  ((has-internal-method :value web-services-which-solve-goal-internal-method)
   (has-capability :value find-web-services-for-goal-web-service-capability)))

(def-class find-web-services-for-goal-web-service-capability (capability)
  ((used-mediator :value find-web-services-for-goal-mediator!))
Appendix D  IRS-III Internal components for SWS

Orchestration

This appendix contains implementation code of IRS-III internal components related to
the orchestration. In particular in contains the code of the implemented orchestration
primitives.

(def-class meta-problem-solving-pattern () ?x
  :iff-def (or ( ?x problem-solving-pattern)
               (= ?x problem-solving-pattern)))

(def-class problem-solving-pattern ()
  ((has-problem-solving-pattern :type unary-procedure)))

(def-class meta-orchestration () ?x
  :iff-def (or ( ?x orchestration)
               (= ?x orchestration)))

(def-class orchestration ()
  ((has-problem-solving-pattern :type problem-solving-pattern)))

(def-relation wsmo-web-service-orchestration-body2 (?web-service ?body)
  :sufficient
  (or (instance ?web-service)
      (wsmo-web-service-orchestration-body2
       (the-parent ?web-service) ?body)
      (and (meta-web-service ?web-service)
           (or (and ( ?interface
                      (the-class-slot-value ?web-service has-interface))
                (meta-interface ?interface)
                (= ?orchestration
                   (the-class-slot-value ?interface has-orchestration))
                (meta-orchestration ?orchestration)
                (<?problem-solving-pattern
                   (the-class-slot-value
                    ?orchestration has-problem-solving-pattern))
                (meta-problem-solving-pattern ?problem-solving-pattern
                   ?body (the-class-slot-value
                          ?problem-solving-pattern has-body)))
           (and (direct-subclass-of ?web-service ?web-service-super)
                (wsmo-web-service-orchestration-body2
                 ?web-service-super ?body)))))

(def-relation goal-associated-mediator (?goal ?mediators)
  "Gets the mediators used by a goal"
  :sufficient
  (or (instance ?goal)
      (goal-associated-mediator (the-parent ?goal)
                                 ?mediators)
      (and (class ?goal)
           (meta-goal ?goal goal)
           (- ?mediators
              (append
               (all-class-slot-values ?goal used-mediator)
               (setofall ?m
                        (meta-mediator ?m)
                        (<?goal
                         (the-class-slot-value
                          ?m has-target-component)))))))))

278
(def-relation wsmo-mediator-source (?mediator ?source)
"Gets the source associated with a mediator"
:sufficient
(or (and (instance ?mediator)
 (wsmo-mediator-source (the-parent ?mediator) ?source))
 (and (meta-mediator ?mediator)
 (equality ?source (the-class-slot-value
 ?mediator has-source-component)))))

(def-relation wsmo-mediator-target (?mediator ?target)
"Gets the target associated with a mediator"
:sufficient
(or (and (instance ?mediator)
 (wsmo-mediator-target (the-parent ?mediator) ?source))
 (and (meta-mediator ?mediator)
 (equality ?target (the-class-slot-value
 ?mediator has-target-component)))))

;;; Some orchestration primitives
(def-procedure set-goal-value (?instance ?slot ?value)
 :body (do
 (unassert (list-of ?slot ?instance 'any))
 (tell (list-of ?slot ?instance ?value))))

(def-function orch-get-goal-value (?goal-name)
:lispy-fun
#'(lambda (goal-name)
 (get-last-achieve-goal-value goal-name)))

(def-function orch-get-all-goal-values (?goal-name)
:lispy-fun
#'(lambda (goal-name)
 (get-all-achieve-goal-values goal-name)))

;;; Manages instance exchanges between the server and publisher
(def-function orch-get-instance-name (?inst ?ontology)
:lispy-fun
#'(lambda (inst ontology)
 (select-ontology ontology)
 (generate-ocml-instances-source
 (cons (first inst) (ocml::slot-values (first inst) 'has-objects)))))

(def-function orch-get-instance-value (?inst-list ?ontology)
:lispy-fun
#'(lambda (inst-list ontology)
 (select-ontology ontology)
 (eval-ocml-instance (first (edr inst-list)))
 (car inst-list)))

(def-function prog (srest forms)
:lispy-fun
#'(lambda (srest forms)
 (car (last forms))))

(def-function orch-seq (&rest ?goals) -> ?sequence-result
:lispy-fun
#'(lambda (?rest goals)
 (internal-sequence-execution goals)))

(def-function achieve-goal (?goal-type &rest ?goal-values)
:lispy-fun
#'(lambda (?goal-type &rest goal-values)
 (internal-achieve-goal goal-type goal-values)))
Appendix E  The Business Process Modelling Ontology

This appendix contains the Business Process Modelling Ontology – BPMO (http://ip-super.org/ontologies/process/bpmo/v2.0.1#bpmo). We present a visualization of BPMO in E.1 and the ontology in WSML in Section E.2.

E.1 BPMO visualization
E.2 BPMO

wsmlVariant _http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _http://ip-super.org/ontologies/process/bpmo/v2.0.1#"*
  upo _http://ip-super.org/ontologies/process/upo/v2.0.1#",
  dc _http://purl.org/dc/elements/1.1#",
  wsmostudio _http://www.wsmostudio.org#" }

ontology bpmo
nonFunctionalProperties
dc#description hasValue "European FP6 Integrated Project SUPER"
wsmostudio#version hasValue "0.7.3.2"
dc#contributor hasValue "Joerg Nitzsche"
dc#description hasValue "Semantic Business Process Management"
dc#description hasValue "[Liliana Cabral", "Roxana Belecheanu", "Barry Norton"]"
dc#description hasValue "WSML Model of the Business Process Modelling Ontology"
dc#description hasValue "en-GB"
dc#date hasValue "SDate: 03/10/2008"
dc#description hasValue "BPMO - Business Process Modelling Ontology v2.0.1"
endNonFunctionalProperties

importsOntology _http://ip-super.org/ontologies/process/upo/v2.0.1#upo"

concept BusinessActivityNonFunctionalProperties
nonFunctionalProperties
dc#description hasValue "The set of non-functional properties of a Process or Task."
endNonFunctionalProperties

concept BusinessDomain subConceptOf upo#BusinessDomain

concept BusinessFunction subConceptOf upo#BusinessFunction

concept BusinessStrategy subConceptOf upo#BusinessStrategy

concept BusinessPolicy subConceptOf upo#BusinessPolicy

concept BusinessProcessMetrics subConceptOf upo#BusinessProcessMetrics

concept BusinessResource subConceptOf upo#Resource

concept BusinessProcessGoal subConceptOf upo#BusinessProcessGoal

concept BusinessActor subConceptOf upo#Agent
nonFunctionalProperties
dc#description hasValue "Identifies an actor."
endNonFunctionalProperties

concept BusinessRole subConceptOf upo#Role
nonFunctionalProperties
dc#description hasValue "The role played by a partner in a process interaction."
endNonFunctionalProperties

281
concept BusinessActivity subConceptOf upo#BusinessActivity
  nonFunctionalProperties
    dc#description hasValue "A Process or Task, typically the result of a business process design or business process engineering activity."
  endNonFunctionalProperties
  hasName ofType (0 1) string
  hasDescription ofType (0 1) string
  hasNonFunctionalProperties ofType (0 1) BusinessActivityNonFunctionalProperties
  hasBusinessDomain ofType upo#BusinessDomain
  hasBusinessFunction ofType upo#BusinessFunction
  hasBusinessStrategy ofType upo#BusinessStrategy
  hasBusinessPolicy ofType upo#BusinessPolicy
  hasBusinessProcessMetrics ofType upo#BusinessProcessMetrics
  hasBusinessProcessGoal ofType upo#BusinessProcessGoal
  hasBusinessResource ofType upo#Resource

concept Process subConceptOf { BusinessActivity, upo#BusinessProcessModel}
  nonFunctionalProperties
    dc#description hasValue "Business process as an identifiable set of tasks, decision points and their associated workflow."
  endNonFunctionalProperties

concept Workflow subConceptOf upo#ProcessOrchestrationSpecification
  nonFunctionalProperties
    dc#description hasValue "Upper class for defining the container of a Process workflow."
  endNonFunctionalProperties
  hasHomeProcess ofType (0 1) Process
  hasFirstWorkflowElement ofType (1 I) WorkflowElement

concept WorkflowElement subConceptOf upo#ProcessOrchestrationElement
  hasHomeProcess ofType (0 1) Process

concept SubProcess subConceptOf { WorkflowElement, Process}

concept Task subConceptOf { BusinessActivity, WorkflowElement, upo#AtomicBusinessActivity}
  nonFunctionalProperties
    dc#description hasValue "An atomic unit of work, i.e. it cannot be described in terms of other workflow entities."
  endNonFunctionalProperties

concept MediationTask subConceptOf Task
  nonFunctionalProperties
    dc#description hasValue "A task with a data mediation purpose"
  endNonFunctionalProperties
  hasSourceTask ofType (0 1) Task
  hasTargetTask ofType (0 1) Task
  hasDataMediator ofType DataMediator

concept GoalTask subConceptOf Task
  nonFunctionalProperties
    dc#description hasValue "A Task for synchronous interaction through a Goal or Web Service."
  endNonFunctionalProperties
  hasPartnerGoal ofType (0 1) SemanticCapability
  hasPartnerRole ofType (0 1) BusinessRole
  messageTo ofType (0 1) Receive
  messageFrom ofType (0 1) Send
  hasInputDescription ofType SemanticCapability
  hasOutputDescription ofType SemanticCapability
  requestsCapability ofType (0 1) SemanticCapability
  providesCapability ofType (0 1) SemanticCapability

concept Send subConceptOf Task
  nonFunctionalProperties
    dc#description hasValue "A Task for asynchronous interaction. Indicates sending a message."
  endNonFunctionalProperties
  hasPartnerWebService ofType (0 1) SemanticCapability
  hasPartnerRole ofType (0 1) BusinessRole
  hasReceiveCounterpart ofType (0 1) Receive
  messageTo ofType (0 1) Receive
hasOutputDescription ofType SemanticCapability
requestsCapability ofType (0 1) SemanticCapability

concept Receive subConceptOf Task
nonFunctionalProperties
dc:description hasValue "A Task for asynchronous interaction. Indicates receiving a message." endNonFunctionalProperties
hasPartnerWebService ofType (0 1) SemanticCapability
hasPartnerRole ofType (0 1) BusinessRole
hasSendCounterpart ofType Send
hasInputDescription ofType SemanticCapability
providesCapability ofType (0 1) SemanticCapability

concept ReceiveMessageEvent subConceptOf { IntermediateEvent, Receive}
nonFunctionalProperties
dc:description hasValue "An event and a Receive Task. Indicates receiving a message." endNonFunctionalProperties

concept WorkflowEvent subConceptOf WorkflowElement
hasName ofType (0 1)_string
hasDescription ofType (0 1)_string

concept StartEvent subConceptOf WorkflowEvent

concept EndEvent subConceptOf WorkflowEvent

concept IntermediateEvent subConceptOf WorkflowEvent

concept TimerEvent subConceptOf IntermediateEvent
nonFunctionalProperties
dc:description hasValue "Signals the receiving of a time-out. Shows the time" endNonFunctionalProperties
hasTimer ofType dateTime

concept ErrorEvent subConceptOf IntermediateEvent
nonFunctionalProperties
dc:description hasValue "Signals the receiving of an error. Shows the error message" endNonFunctionalProperties
hasErrorMessage ofType string

concept SemanticCapability
nonFunctionalProperties
dc:description hasValue "Provides a pointer (e.g. uri) to a semantic Description; for instance a WSMO Goal or Web Service" endNonFunctionalProperties
hasSemanticDescription ofType (0 l)_string

concept Branch
nonFunctionalProperties
dc:description hasValue "A workflow branch, which contains workflow elements " endNonFunctionalProperties
hasElement ofType (0 1) WorkflowElement

concept OrderedElement subConceptOf Branch
hasOrder ofType (0 1) _integer
hasNextElement ofType (0 1) OrderedElement

concept ConditionalBranch subConceptOf Branch
nonFunctionalProperties
dc:description hasValue "A branch of workflow executed depending on the satisfaction of a condition, or unconditionally." endNonFunctionalProperties
hasCondition ofType (0 1) Condition
hasOrder ofType (0 1) _integer
hasNextElement ofType (0 1) ConditionalBranch

concept UnconditionalBranch subConceptOf Branch
nonFunctionalProperties
dc:description hasValue "A branch of a workflow with no condition."
Appendix E

concept EventBranch subConceptOf Branch
hasEvent ofType (1 1) IntermediateEvent

concept DeferredChoiceMerge subConceptOf BlockPattern
nonFunctionalProperties
  dc#description hasValue "A pair of deferred choice and simple merge gateways."
endNonFunctionalProperties
hasEventBranch ofType (2 *) EventBranch

concept ExclusiveChoiceMerge subConceptOf BlockPattern
nonFunctionalProperties
  dc#description hasValue "A pair of exclusive choice and exclusive merge gateways."
endNonFunctionalProperties
hasSize ofType (0 1) integer
hasConditionalBranch ofType (2 *) ConditionalBranch
hasDefaultBranch ofType (0 1) UnconditionalBranch

concept MultipleChoiceMultiMerge subConceptOf BlockPattern
nonFunctionalProperties
  dc#description hasValue "A pair of multiple choice and exclusive merge gateways."
endNonFunctionalProperties
hasSize ofType (0 1) integer
hasConditionalBranch ofType (2 *) ConditionalBranch
hasDefaultBranch ofType (0 1) UnconditionalBranch

concept MultipleChoiceDiscriminator subConceptOf BlockPattern
nonFunctionalProperties
  dc#description hasValue "A pair of multiple choice and exclusive discriminator gateways."
endNonFunctionalProperties
hasSize ofType (0 1) integer
hasConditionalBranch ofType (2 *) ConditionalBranch

concept ParallelSplitSynchronise subConceptOf BlockPattern
nonFunctionalProperties
  dc#description hasValue "A pair of parallel split and synchronise gateways."
endNonFunctionalProperties
hasBranch ofType (2 *) UnconditionalBranch

concept ParallelSplitDiscriminator subConceptOf BlockPattern
nonFunctionalProperties
  dc#description hasValue "A pair of parallel split and discriminator gateways."
endNonFunctionalProperties
hasBranch ofType (2 *) UnconditionalBranch

concept BlockPattern subConceptOf WorkflowElement
nonFunctionalProperties
  dc#description hasValue "A component with one source object, one sink object, one outside entry point and one outside exit point (see Definition 3, page 6, in (Ouyang et al.))"
endNonFunctionalProperties

concept Sequence subConceptOf BlockPattern
nonFunctionalProperties
  dc#description hasValue "A list of workflow entities"
endNonFunctionalProperties
hasOrderedElement ofType (1 *) OrderedElement
hasSize ofType (0 1) integer

concept Loop subConceptOf BlockPattern
hasCondition ofType (0 1) Condition
executes ofType (0 1) WorkflowElement

concept GraphPattern subConceptOf WorkflowElement

concept DeferredChoice subConceptOf OutgoingFlow
nonFunctionalProperties
  dc#description hasValue "A point in the workflow where one of several branches is chosen, based on input from the environment (e.g. client)"
endNonFunctionalProperties
Appendix E

hasOutgoingConnector ofType (2 *) ControlFlowConnector

concept ExclusiveChoice subConceptOf OutgoingFlow
nonFunctionalProperties
  dc#description hasValue "A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen."
  endNonFunctionalProperties
hasSize ofType (0 1) integer
hasOutgoingConnector ofType (2 *) ConditionalFlowConnector
hasDefaultOutgoingConnector ofType (0 1) ControlFlowConnector

concept ParallelSplit subConceptOf OutgoingFlow
nonFunctionalProperties
  dc#description hasValue "A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order."
  endNonFunctionalProperties
hasOutgoingConnector ofType (2 *) ControlFlowConnector

concept MultipleChoice subConceptOf OutgoingFlow
  hasSize ofType (0 1) integer
  hasOutgoingConnector ofType (2 *) ConditionalFlowConnector

concept InterleavedParallelRouting subConceptOf OutgoingFlow
  hasOutgoingConnector ofType (2 *) ControlFlowConnector

concept SimpleMerge subConceptOf Merge
nonFunctionalProperties
  dc#description hasValue "A point in the workflow process where two or more alternative branches come together without synchronization. None of the alternative branches is ever executed in parallel."
  endNonFunctionalProperties

concept Synchronisation subConceptOf Merge
nonFunctionalProperties
  dc#description hasValue "A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once."
  endNonFunctionalProperties

concept MultiMerge subConceptOf Merge
nonFunctionalProperties
  dc#description hasValue "A point in a workflow process where two or more branches reconverge without synchronization. Branches could be activated concurrently."
  endNonFunctionalProperties

concept MultipleMergeSynchronise subConceptOf Merge
nonFunctionalProperties
  dc#description hasValue "A point in a workflow process where two or more branches reconverge with synchronization. Branches could be activated concurrently."
  endNonFunctionalProperties

concept Discriminator subConceptOf Merge
nonFunctionalProperties
  dc#description hasValue "A point in a workflow process where two or more branches reconverge with Discriminator. Branches could be activated concurrently."
  endNonFunctionalProperties

concept Condition
  hasExpression ofType (0 1) string

concept Repeat subConceptOf Loop
nonFunctionalProperties
  dc#description hasValue "A loop which executes a workflow entity at least once."
  endNonFunctionalProperties

concept While subConceptOf Loop
nonFunctionalProperties
  dc#description hasValue "A loop which executes a workflow entity 0 or more times, depending on a condition."
  endNonFunctionalProperties

concept ControlFlowPattern subConceptOf GraphPattern

285
Appendix E

concept OutgoingFlow subConceptOf ControlflowPattern

class IncomingFlow subConceptOf ControlflowPattern

class Merge subConceptOf IncomingFlow

hasIncomingConnector ofType (2 *) ControlflowConnector

concept ControlflowConnector subConceptOf ControlflowPattern

hasSource ofType (1 1) WorkflowElement

hasTarget ofType (1 1) WorkflowElement

class ConditionalflowConnector subConceptOf ControlflowConnector

hasCondition ofType (0 1) Condition

hasOrder ofType (0 1) _integer

concept ProcessFragment subConceptOf BusinessActivity

nonFunctionalProperties

dc#description hasValue "The concept of a Process Fragment."
endNonFunctionalProperties

hasName ofType (0 1) _string

hasDescription ofType (0 1) _string

hasWorkflowElements ofType WorkflowElement

hasStartElement ofType (1 1) WorkflowElement

hasEndElement ofType (1 1) WorkflowElement

belongsToProcess ofType (1 1) Process

concept Mediator subConceptOf upo#BusinessProcessMediator

hasName ofType (0 1) _string

hasDescription ofType (0 1) _string

concept ProcessMediator subConceptOf Mediator

hasSourceProcess ofType (1 1) Process

hasTargetProcess ofType (1 *) Process

hasMediationProcess ofType (0 1) MediationProcess

hasSWSMediator ofType (0 1) SemanticCapability

concept DataMediator subConceptOf Mediator

hasMediator ofType SemanticCapability

hasMediationService ofType SemanticCapability

hasInputDescription ofType (0 1) SemanticCapability

hasOutputDescription ofType (0 1) SemanticCapability

concept MediationProcess subConceptOf Process

E.3 SBPEL

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://ip-super.org/ontologies/process/sbpel/v2.0.0#",

sbpel _"http://ip-super.org/ontologies/process/sbpel/v2.0.0#",

dc _"http://purl.org/dc/Elements/l.1#",
bpmo _"http://ip-super.org/ontologies/process/bpmo/v2.0.1#",
upo _"http://ip-super.org/ontologies/process/upo/v2.0.1#",

bpel _"http://ip-super.org/ontologies/process/bpel20/v2.0.0#"}

ontology sbpel

nonFunctionalProperties

hasValue "sbpel Ontology"

dc#description hasValue "WSML Model of the semantic Extensions to BPEL 2.0"

dc#publidate hasValue "SUPER European Integrated Project"

dc#publication hasValue ("bpel", "Business Process", "workFlow")
concept SemanticProcess subConceptOf bpel#Process
  nonFunctionalProperties
   escription hasValue "Concept of being a <process>-Element of an executable sbpel Process"
  endNonFunctionalProperties
  hasConversation ofType Conversation
  hasPartner ofType Partner
  hasSemanticOnMessage ofType SemanticOnMessage

concept SemanticScope subConceptOf bpel#Scope
  nonFunctionalProperties
   escription hasValue "Concept of being an Scope used within a semantic process"
  endNonFunctionalProperties
  hasSemanticOnMessage ofType SemanticOnMessage

concept Partner
  nonFunctionalProperties
   escription hasValue "Concept of being a <partner>-Element"
  endNonFunctionalProperties
  hasName ofType (1) string
  hasBusinessEntity ofType (0 1) _string /* http://www.wsmo.org/wsml/wsml-syntax#Concept */
  belongsToConversation ofType (1) Conversation
  hasInputVariable ofType (1) SemanticVariable
  hasOutputVariable ofType (1) SemanticVariable

concept SendReceive subConceptOf {bpel#Interaction, bpel#NewActivityType}
  nonFunctionalProperties
   escription hasValue "Concept of being an activity that only completes after it first sends a message and then receives a message"
  endNonFunctionalProperties
  belongsToConversation ofType (1) Conversation
  hasInputVariable ofType (1) SemanticVariable
  hasOutputVariable ofType (1) SemanticVariable

concept Receive subConceptOf {bpel#Interaction, bpel#NewActivityType}
  nonFunctionalProperties
   escription hasValue "Concept of being an activity that receives a message" 
  endNonFunctionalProperties
  belongsToConversation ofType (1) Conversation
  hasVariable ofType (1) SemanticVariable

concept Send subConceptOf {bpel#Interaction, bpel#NewActivityType}
  nonFunctionalProperties
   escription hasValue "Concept of being an activity that sends a message" 
  endNonFunctionalProperties
  belongsToConversation ofType (1) Conversation
  hasVariable ofType (1) SemanticVariable

concept SemanticPick subConceptOf bpel#NewActivityType
  nonFunctionalProperties
   escription hasValue "Concept of being a semantic Pick"
  endNonFunctionalProperties
  doesCreateInstance ofType (0 1) _boolean
  hasOnMessage ofType (1 *) bpel#MessageEvent
hasOnAlarm ofType bpel#OnAlarm

concept SemanticOnMessage subConceptOf {bpel#Interaction, bpel#MessageEvent}
nonFunctionalProperties
  dc#description hasValue "Concept of being a semantic onMessage"
endNonFunctionalProperties
belongsToConversation ofType (1) Conversation
hasVariable ofType (1) SemanticVariable
hasActivity ofType (1) bpel#Activity

concept Conversation
nonFunctionalProperties
  dc#description hasValue "Concept of being a Conversation"
endNonFunctionalProperties

hasVariable ofType (1) string
describesInterface ofType (1) InterfaceDescription

concept InterfaceDescription
nonFunctionalProperties
  dc#description hasValue "Concept of describing what kind of interface the Conversation defines"
endNonFunctionalProperties

hasName ofType (1) string

concept IncomingInterface subConceptOf InterfaceDescription
nonFunctionalProperties
  dc#description hasValue "Concept of being an incoming interface"
endNonFunctionalProperties

hasWebServiceDescription ofType (1) _string /*_"http://www.wsmo.org/wsmi/wsml-syntax#Webservice"*/

concept OutgoingInterface subConceptOf InterfaceDescription
nonFunctionalProperties
  dc#description hasValue "Concept of being an outgoing interface"
endNonFunctionalProperties

hasGoalDescription ofType (1) _string /*_"http://www.wsmo.org/wsmi/wsml-syntax#Goal"*/

concept sbpel#Mediate subConceptOf bpel#ExtensionAssignOperation
nonFunctionalProperties
  dc#description hasValue "Concept of being an <extensionAssignOperation>-Element"
endNonFunctionalProperties

hasInputVariable ofType (1) SemanticVariable
hasOutputVariable ofType (1) SemanticVariable
usesDataMediator ofType (0 1) DataMediator

correspondsTo ofType bpmo#DataMediator

concept DataMediator subConceptOf upo#BusinessProcessMediator

nonFunctionalProperties
  dc#description hasValue "Concept of being a (WSMO) data mediator"
endNonFunctionalProperties

sbpel#usesMediator ofType _string

hasMediationService ofType _string corrspondsTo ofType bpmo#DataMediator

namespace {_"http://ip-super.org/ontologies/process/bpel20/v2.0.0#",
  bpel "http://ip-super.org/ontologies/process/bpel20/v2.0.0#",
  dc "http://purl.org/dc/Elements/1.1#",
  wsdix "http://ip-super.org/ontologies/process/WSDLExtension4BPEL/v2.0.0#",
  upo "http://ip-super.org/ontologies/process/upo/v2.0.1#",
  bpmo "http://ip-super.org/ontologies/process/bpmo/v2.0.1#"}

ontology bpel20

288
APPENDIX E

nonFunctionalProperties
  hasValue "BPEL Ontology"
  dc#description hasValue "WSML Model of the BPEL 2.0 metamodel"
  hasValue "SUPER European Integrated Project"
  hasValue ["BPEL", "Business Process", "workFlow"]
  hasValue "Joerg Nitzsche"
  hasValue ["Joerg Nitzsche", "Barry Norton", "Taimo van Lessen", "Daniel Wutke"]
  hasValue "en-US"
  hasValue "SDatc: 2008/10/14"
endNonFunctionalProperties

importsOntology
  {"http://ip-super.org/ontologies/process/WSDLExtension4BPEL/v2.0.0#WSDLExtension4BPEL",
   "http://ip-super.org/ontologies/process/bpmo/v2.0.1#bpmo"}

axiom allowedInitiateValues
  definedBy !- ?x[doesInitiate hasValue ?initiate] memberOf Correlation
  and ?initiate != yes
  and ?initiate != no
  and ?initiate != bpel#join.

axiom allowedPatternValues
  definedBy !- ?x[hasPattern hasValue ?pattern] memberOf Correlation
  and ?pattern != response
  and ?pattern != request
  and ?pattern != _"http://ip-super.org/ontologies/process/bpel20/v2.0.0#request-response".

axiom allowedEndpointReferenceValues
  definedBy !- ?x[hasEndpointReference hasValue ?value] memberOf CopyPartnerLinkEndpointReference
  and ?value != myRole
  and ?value != partnerRole.

axiom disjointFaultHandlers
  definedBy !- ?x memberOf Catch
  and ?x memberOf CatchAll.

axiom disjointWaitStatements
  definedBy !- ?x memberOf For
  and ?x memberOf bpel#Until.

axiom disjointDataTypes
  definedBy !- ?x memberOf Type
  and ?x memberOf MessageType or ?x memberOf Element or ?x memberOf Element and ?x memberOf MessageType.

axiom disjointActivityTypes
  definedBy !- ?x memberOf BasicActivity
  and ?x memberOf StructuredActivity.

instance yes memberOf Initiate
instance no memberOf Initiate
concept Initiate
instance bpel#join memberOf Initiate
concept Pattern
instance response memberOf Pattern
instance request memberOf Pattern
instance _"http://ip-super.org/ontologies/process/bpel20/v2.0.0#request-response" memberOf Pattern
concept EndpointReference

289
instance myRole memberOf EndpointReference

instance partnerRole memberOf EndpointReference

concept Process subConceptOf upo#BusinessProcessModel
  nonFunctionalProperties
  xmlns hasValue "http://docs.oasis-open.org/wsBPEL/2.0/Process/executable"
  dc#description hasValue "Concept of being a <process>-Element of an executable BPEL Process"
  endNonFunctionalProperties
  hasName ofType (1) _string
  hasTargetNamespace ofType (1) _string /* _iri */
  hasQueryLanguage ofType (0 1) _string /* _iri */
  hasExpressionLanguage ofType (0 1) _string /* _iri */
  doesSuppressJoinFailure ofType (0 1) _boolean
  hasExtension ofType Extension
  hasImport ofType Import
  hasPartnerLink ofType bpel#PartnerLink
  hasMessageExchange ofType MessageExchange
  hasVariable ofType Variable
  hasCorrelationSet ofType CorrelationSet
  hasCatch ofType Catch
  hasCatchAll ofType (0 1) CatchAll
  hasOnEvent ofType OnEvent
  hasOnAlarm ofType RepeatableOnAlarm
  hasActivity ofType (1) Activity
  correspondsTo ofType bpmo#Process

concept Extension
  nonFunctionalProperties
  dc#description hasValue "Concept of being a <extension>-Element"
  endNonFunctionalProperties
  hasNamespace ofType (1) _string /* _iri */
  isMustUnderstand ofType (1) _boolean

concept Import
  nonFunctionalProperties
  dc#description hasValue "Concept of being a <import>-Element"
  endNonFunctionalProperties
  hasNamespace ofType (1) _string /* _iri */
  hasLocation ofType (0 1) _string /* _iri */
  hasImportType ofType (1) _string /* _iri */

concept bpel#PartnerLink
  nonFunctionalProperties
  dc#description hasValue "Concept of being a <partnerLink>-Element"
  endNonFunctionalProperties
  hasName ofType (1) _string
  hasPartnerLinkType ofType (1) wsdlx#PartnerLinkType
  hasMyRole ofType (0 1) wsdlx#Role
  hasPartnerRole ofType (0 1) wsdlx#Role
  doesInitializePartnerRole ofType (0 1) _boolean

concept Variable
  nonFunctionalProperties
  dc#description hasValue "Concept of being a <variable>-Element"
  endNonFunctionalProperties
  hasName ofType (1) _string
  hasType ofType (1) Type
  hasFromSpecification ofType (0 1) FromSpecification

concept Type
  nonFunctionalProperties
  dc#description hasValue "Concept of representing data types"
  endNonFunctionalProperties
  hasDefinition ofType (1) _string /* WSDL fragment/element identifier */

concept XSDType subConceptOf Type
  nonFunctionalProperties
**Appendix E**

```
dc#description hasValue "Concept of representing XSD simple types and XSD complex types"
endNonFunctionalProperties
/* hasDefinition —> WSDL fragment/element identifier: xmlns(namespacedeclaration)wsdl.typeDefinition(type) (see http://www.w3.org/TR/wsdl20/#ffag-ids)*/
/* Example:
http://example.org/TicketAgent.wsdl#xmlns(ns1=http://example.org/TicketAgent.xsd)wsdl.typeDefinition(ns1:listFlightsRequest) */

concept XSDElement subConceptOf Type
nonFunctionalProperties
dc#description hasValue "Concept of being an XSD element"
endNonFunctionalProperties
/* hasDefinition —> WSDL fragment/element identifier: wsdl.elementDeclaration(element) (see http://www.w3.org/TR/wsdl20/#frag-ids)*/
/* Example:
http://example.org/TicketAgent.wsdl#xmlns(ns1=http://example.org/TicketAgent.xsd)wsdl.elementDeclaration(ns1:listFlightsRequest) */

concept WSDLMessageType subConceptOf Type
nonFunctionalProperties
dc#description hasValue "Concept of being a WSDL message type"
endNonFunctionalProperties
/* hasDefinition —> WSDL fragment/element identifier: wsdl1.message(message) (see http://www.w3.org/TR/wsdl11elementidentifiers/)*/

concept CorrelationSet
nonFunctionalProperties
dc#description hasValue "Concept of being a <correlationSet>-Element"
endNonFunctionalProperties

concept CompensationHandler
nonFunctionalProperties
dc#description hasValue "Concept of being a <compensationHandler>-Element"
endNonFunctionalProperties
hasActivity ofType (1) Activity

concept FaultHandler
nonFunctionalProperties
dc#description hasValue "Concept of being a faultHandler"
endNonFunctionalProperties
hasActivity ofType (1) Activity

concept Catch subConceptOf FaultHandler
nonFunctionalProperties
dc#description hasValue catchDisjointCatchAll
endNonFunctionalProperties
hasFaultName ofType (0 1)_string /* WSDL fragment/element identifier: wsdl11.portType/operation.fault(portType/operation/fault) (see http://www.w3.org/TR/wsdl11elementidentifiers/)*/
hasFaultVariable ofType (0 1) Variable
hasFaultType ofType (0 1) Type

concept CatchAll subConceptOf FaultHandler
nonFunctionalProperties
dc#description hasValue catchDisjointCatchAll
endNonFunctionalProperties

concept TerminationHandler
nonFunctionalProperties
dc#description hasValue "Concept of being a <terminationHandler>-Element"
endNonFunctionalProperties
hasActivity ofType (1) Activity

concept OnEvent subConceptOf OnMessage
nonFunctionalProperties
dc#description hasValue "Concept of being an <onEvent>-Element"
```

291
Appendix E

EndNonFunctionalProperties
hasTypeOfMessage ofType (0 1) Type
hasActivity ofType (1) Scope

concept Event
nonFunctionalProperties
dct#description hasValue "Concept of being an Event"
endNonFunctionalProperties

concept MessageEvent subConceptOf Event
nonFunctionalProperties
dct#description hasValue "Concept of being a message Event"
endNonFunctionalProperties

concept OnMessage subConceptOf (WSDLInteraction, MessageEvent)
nonFunctionalProperties
dct#description hasValue "Concept of being an <onMessage>-Element"
endNonFunctionalProperties
hasVariable ofType (0 1) Variable
hasMessageExchange ofType (0 1) MessageExchange
hasFromParts ofType (0 1) FromParts
hasActivity ofType (1) Activity

concept OnAlarm subConceptOf Event
nonFunctionalProperties
dct#description hasValue "Concept of being an <onAlarm>-Element within a <pick>-Activity"
endNonFunctionalProperties
hasWaitStatement ofType (1) WaitStatement
hasActivity ofType (1) Activity

concept RepeatableOnAlarm subConceptOf OnAlarm
nonFunctionalProperties
dct#description hasValue "Concept of being an <onAlarm>-Element"
endNonFunctionalProperties
hasRepeatEvery ofType (0 1) For
hasActivity ofType (1) Scope

concept StandardAttributes
nonFunctionalProperties
dct#description hasValue "Concept of providing the standard attributes For Activities in BPEL"
endNonFunctionalProperties
hasName ofType (0 l)_string
doesSuppressJoinFailure ofType (0 1) boolean
isTarget ofType Link
hasJoinCondition ofType (0 1) Condition
isSource ofType (0 1) OrderedLink

concept OrderedLink
nonFunctionalProperties
dct#description hasValue "Concept used to specify an ordered list of Activities"
endNonFunctionalProperties
hasLink ofType (1) Link
hasOrderedLink ofType (0 1) OrderedLink

concept Activity subConceptOf upo#BusinessActivity
nonFunctionalProperties
dct#description hasValue "Concept of being a BPEL activity"
endNonFunctionalProperties
correspondsTo ofType bpmo#WorkflowElement

concept BasicActivity subConceptOf Activity
nonFunctionalProperties
dct#description hasValue "Concept of being a basic activity"
endNonFunctionalProperties

concept StructuredActivity subConceptOf Activity
nonFunctionalProperties
dct#description hasValue "Concept of being a structured activity"
endNonFunctionalProperties
concept Link
  nonFunctionalProperties
  dc:Description hasValue "Concept of being a <link>-Element"
endNonFunctionalProperties
hasName ofType (1) _string
hasTransitionCondition ofType (0 1) Condition

concept Condition
  nonFunctionalProperties
  dc:Description hasValue "Concept of being a Condition"
endNonFunctionalProperties
hasExpression ofType (1 1) _string
hasExpressionLanguage ofType (0 1) _string /* iri */

concept Interaction
  nonFunctionalProperties
  dc:Description hasValue "Concept of doing interaction"
endNonFunctionalProperties

concept WSDLInteraction subConceptOf Interaction
  nonFunctionalProperties
  dc:Description hasValue "Concept of doing WSDL dependent interaction"
endNonFunctionalProperties
hasPartnerLink ofType (1) bpel#PartnerLink
hasPortType ofType (0 1) _string /* WSDL fragment/element identifier: wsdll.portType(portType) (see
http://www.w3.org/TR/wsdll1#elementidentifiers/) */
hasOperation ofType (1) _string /* WSDL fragment/element identifier: wsdll1.portTypeOperation(portType/operation) (see
http://www.w3.org/TR/wsdll1#elementidentifiers/) */
hasCorrelation ofType Correlation

concept ExtensionActivity subConceptOf BasicActivity
  nonFunctionalProperties
  dc:Description hasValue "Concept of being an ExtensionActivity"
endNonFunctionalProperties
hasActivity ofType (1) NewActivityType

concept NewActivityType subConceptOf StandardAttributes
  nonFunctionalProperties
  dc:Description hasValue "Concept of being a new activity type, i.e. by inheriting from this concept a new activity type with new operational semantics can be defined using another namespace"
endNonFunctionalProperties

concept CorrelationWithPattern subConceptOf Correlation
  nonFunctionalProperties
  dc:Description hasValue "Concept of being a <correlation>-Element"
endNonFunctionalProperties
hasPattern ofType (0 1) Pattern

concept Correlation
  nonFunctionalProperties
  dc:Description hasValue "Concept of being a <correlation>-Element"
endNonFunctionalProperties
hasSet ofType (1) CorrelationSet
doesInitiate ofType (0 1) Initiate

concept Validate subConceptOf BasicActivity
  nonFunctionalProperties
  dc:Description hasValue "Concept of being a <validate>-Activity"
endNonFunctionalProperties
hasVariable ofType (1 *) Variable

concept bpel#Receive subConceptOf {WSDLInteraction, BasicActivity, StandardAttributes}
  nonFunctionalProperties
  dc:Description hasValue "Concept of being a <receive>-Activity"
endNonFunctionalProperties
hasVariable ofType (0 1) Variable
doesCreateInstance ofType (0 1) boolean
hasMessageExchange ofType (0 1) MessageExchange
hasFromParts ofType FromParts

concept MessageExchange
nonFunctionalProperties
dc#description hasValue "Concept of being a MessageExchange attribute"
endNonFunctionalProperties
hasName ofType (1) _string

concept Reply subConceptOf {WSDLInteraction, BasicActivity, StandardAttributes}
nonFunctionalProperties
dc#description hasValue "Concept of being a <reply>-Activity"
endNonFunctionalProperties
hasVariable ofType (1) Variable
hasFaultName ofType (0 1) string /* WSDL fragment/element identifier: wsdl11.portType/operation/fault[portType/operation/fault] (see http://www.w3.org/TR/wsdl11#elementIdentifiers/)*/
hasMessageExchange ofType (0 1) MessageExchange
hasToParts ofType ToParts

concept Invoke subConceptOf {WSDLInteraction, BasicActivity, StandardAttributes}
nonFunctionalProperties
dc#description hasValue "Concept of being an <invoke>-Activity"
endNonFunctionalProperties
hasInputVariable ofType (1) Variable
hasOutputVariable ofType (0 1) Variable
hasCatch ofType Catch
hasCatchAll ofType (0 1) CatchAll
hasCompensationHandler ofType (0 1) CompensationHandler
hasToParts ofType ToParts
hasFromParts ofType FromParts

concept ToParts
nonFunctionalProperties
dc#description hasValue "Concept of being a <toParts>-Element"
endNonFunctionalProperties
hasToPart ofType (1) _string /* WSDL fragment/element identifier: wsdl11.messagePart/message/part) (see http://www.w3.org/TR/wsdl11#elementIdentifiers/)*/
hasFromVariable ofType (1) Variable

concept FromParts
nonFunctionalProperties
dc#description hasValue "Concept of being a <fromParts>-Element"
endNonFunctionalProperties
hasFromPart ofType (1) _string /* WSDL fragment/element identifier: wsdl11.messagePart/message/part) (see http://www.w3.org/TR/wsdl11#elementIdentifiers/)*/
hasToVariable ofType (1) Variable

concept Assign subConceptOf {BasicActivity, StandardAttributes}
nonFunctionalProperties
dc#description hasValue "Concept of being an <assign>-Activity"
endNonFunctionalProperties
hasValidate ofType (0 1) boolean
hasAssignOperation ofType (1 *) AssignOperation

concept AssignOperation
nonFunctionalProperties
dc#description hasValue "Concept of being an assign operation"
endNonFunctionalProperties

doesKeepSrcElementName ofType (0 1) boolean
hasFromSpecification ofType (1) CopySpecification
hasToSpecification ofType (1) CopySpecification

hasToPart ofType (1) _string /* WSDL fragment/element identifier: wsdl11.messagePart/message/part) (see http://www.w3.org/TR/wsdl11#elementIdentifiers/)*/
hasFromVariable ofType (1) Variable

concept FromParts
nonFunctionalProperties
dc#description hasValue "Concept of being a <fromParts>-Element"
endNonFunctionalProperties
hasFromPart ofType (1) _string /* WSDL fragment/element identifier: wsdl11.messagePart/message/part) (see http://www.w3.org/TR/wsdl11#elementIdentifiers/)*/
hasToVariable ofType (1) Variable

concept Assign subConceptOf {BasicActivity, StandardAttributes}
nonFunctionalProperties
dc#description hasValue "Concept of being an <assign>-Activity"
endNonFunctionalProperties
hasValidate ofType (0 1) boolean
hasAssignOperation ofType (1 *) AssignOperation

concept AssignOperation
nonFunctionalProperties
dc#description hasValue "Concept of being an assign operation"
endNonFunctionalProperties

doesKeepSrcElementName ofType (0 1) boolean
hasFromSpecification ofType (1) CopySpecification
hasToSpecification ofType (1) CopySpecification

294
concept CopySpecification
  nonFunctionalProperties
    dc#description hasValue "Concept of being a copy specification"
  endNonFunctionalProperties

concept CopyVariablePart subConceptOf CopySpecification
  nonFunctionalProperties
    dc#description hasValue "Concept used to specify which part of a variable should be written or read"
  endNonFunctionalProperties
  hasVariable ofType (1) Variable
  hasPart ofType (0 1) _string /* WSDL fragment/element identifier: wsdl11.messagePart/message/part (see http://www.w3.org/TR/wsdl11elementidentifiers/)*/

concept Query
  nonFunctionalProperties
    dc#description hasValue "Concept used define a query"
  endNonFunctionalProperties
  hasQuery ofType (0 1) Query
  hasQueryLanguage ofType (0 1) _string /* iri */

concept CopyPartnerLinkEndpointReference subConceptOf CopySpecification
  nonFunctionalProperties
    dc#description hasValue "Concept used to specify from which partnerLink the endpointreference should be copied"
  endNonFunctionalProperties
  hasPartnerLink ofType (1) bpel#PartnerLink
  hasEndpointReference ofType (1) EndpointReference

concept CopyVariableProperty subConceptOf CopySpecification
  nonFunctionalProperties
    dc#description hasValue "Concept used to specify which variable has to be written or read"
  endNonFunctionalProperties
  hasVariable ofType (1) Variable
  hasProperty ofType (0 1) _string /* WSDL fragment/element identifier: (see http://www.w3.org/TR/wsdl11elementidentifiers/)*/

concept CopyExpression subConceptOf CopySpecification
  nonFunctionalProperties
    dc#description hasValue "Concept used to specify an expression"
  endNonFunctionalProperties
  hasExpressionLanguage ofType (0 1) _string /* iri */
  hasExpression ofType (1) _string

concept CopyLiteral subConceptOf CopySpecification
  nonFunctionalProperties
    dc#description hasValue "Concept used to specify a literal"
  endNonFunctionalProperties
  hasLiteralValue ofType (1) _string

concept CopyPartnerLink subConceptOf CopySpecification
  nonFunctionalProperties
    dc#description hasValue "Concept used to specify to which partnerLink an endpointreference should be written"
  endNonFunctionalProperties
  hasPartnerLink ofType (1) bpel#PartnerLink

concept Throw subConceptOf {BasicActivity, StandardAttributes}
  nonFunctionalProperties
    dc#description hasValue "Concept of being a <throw>-Activity"
  endNonFunctionalProperties
  hasFaultName ofType (1) _string /* WSDL fragment/element identifier: wsdl11.portType/operation/fault(portType/operation/fault) (see http://www.w3.org/TR/wsdl11elementidentifiers/)*/
  hasFaultVariable ofType (0 1) Variable

concept Wait subConceptOf {BasicActivity, StandardAttributes}
  nonFunctionalProperties
    dc#description hasValue "Concept of being a <wait>-Activity"
  endNonFunctionalProperties
  hasWaitStatement ofType (1) WaitStatement
concept WaitStatement
  nonFunctionalProperties
dc#description hasValue "Concept of being a wait statement"
endNonFunctionalProperties

concept For subConceptOf WaitStatement
  nonFunctionalProperties
    hasValue forDisjointUntil
dc#description hasValue "Concept of being a <For>-Element"
endNonFunctionalProperties
  hasExpressionLanguage ofType (0 1) _string /* iri */
  hasDurationExpression ofType (1) _string

concept bpel#Until subConceptOf WaitStatement
  nonFunctionalProperties
    hasValue forDisjointUntil
dc#description hasValue "Concept of being an <Until>-Element"
endNonFunctionalProperties
  hasExpressionLanguage ofType (0 1) _string /* iri */
  hasDeadlineExpression ofType (1) _string

concept Empty subConceptOf \{BasicActivity, StandardAttributes\}
  nonFunctionalProperties
dc#description hasValue "Concept of being an <empty>-Activity"
endNonFunctionalProperties

concept Exit subConceptOf \{BasicActivity, StandardAttributes\}
  nonFunctionalProperties
dc#description hasValue "Concept of being an <exit>-Activity"
endNonFunctionalProperties

concept Rethrow subConceptOf \{BasicActivity, StandardAttributes\}
  nonFunctionalProperties
dc#description hasValue "Concept of being a <rethrow>-Activity"
endNonFunctionalProperties

concept Compensate subConceptOf \{BasicActivity, StandardAttributes\}
  nonFunctionalProperties
dc#description hasValue "Concept of being an <compensate>-Activity"
endNonFunctionalProperties

concept CompensateScope subConceptOf \{BasicActivity, StandardAttributes\}
  nonFunctionalProperties
dc#description hasValue "Concept of being an <compensateScope>-Activity"
endNonFunctionalProperties
  compensatesScope ofType (1) Scope

concept Sequence subConceptOf \{StructuredActivity, StandardAttributes\}
  nonFunctionalProperties
dc#description hasValue "Concept of being a <sequence>-Activity"
endNonFunctionalProperties
  hasOrderedActivity ofType (1) OrderedActivity

concept OrderedActivity
  nonFunctionalProperties
dc#description hasValue "Concept used to specify an ordered list of Activities"
endNonFunctionalProperties
  hasActivity ofType (1) Activity
  hasOrderedActivity ofType (0 1) OrderedActivity

concept OrderedConditionalBranch
  nonFunctionalProperties
dc#description hasValue "Concept used to specify an ordered list of conditional branches"
endNonFunctionalProperties
  hasIfBranch ofType (1) ConditionalBranch
  hasElseIfBranch ofType (0 1) OrderedConditionalBranch

concept bpel#If subConceptOf \{StructuredActivity, StandardAttributes\}
  nonFunctionalProperties
Appendix E

dc:description hasValue "Concept of being a <if>-Activity"
endNonFunctionalProperties
hasIfBranch ofType (1) ConditionalBranch
hasElseIfBranch ofType OrderedConditionalBranch
hasElseBranch ofType (0 1) bpel#Else

concept ConditionalBranch
nonFunctionalProperties
dc:description hasValue "Concept of being a conditional branch"
endNonFunctionalProperties
hasCondition ofType (1) Condition
hasActivity ofType (1) Activity

concept bpel#Else
nonFunctionalProperties
dc:description hasValue "Concept of being a <else>-branch"
endNonFunctionalProperties
hasActivity ofType (1) Activity

concept bpel#While subConceptOf {StructuredActivity, StandardAttributes}
nonFunctionalProperties
dc:description hasValue "Concept of being a <while>-Activity, i.e. first the Condition is evaluated and then the Activity
is done"
endNonFunctionalProperties
hasCondition ofType (1) Condition
hasActivity ofType (1) Activity

concept RepeatUntil subConceptOf {StructuredActivity, StandardAttributes}
nonFunctionalProperties
dc:description hasValue "Concept of being a <repeatUntil>-Activity, i.e. first the Activity is done and then the Condition is
evaluated"
endNonFunctionalProperties
hasCondition ofType (1) Condition
hasActivity ofType (1) Activity

concept Pick subConceptOf {BasicActivity, StandardAttributes}
nonFunctionalProperties
dc:description hasValue "Concept of being a <pick>-Activity"
endNonFunctionalProperties
doesCreateInstance ofType (0 1) boolean
hasOnMessage ofType (1 *) OnMessage
hasOnAlarm ofType OnAlarm

concept Flow subConceptOf {StructuredActivity, StandardAttributes}
nonFunctionalProperties
dc:description hasValue "Concept of being a <flow>-Activity"
endNonFunctionalProperties
hasLink ofType (1) Link
hasActivity ofType (1 *) Activity

concept ForEach subConceptOf {StructuredActivity, StandardAttributes}
nonFunctionalProperties
dc:description hasValue "Concept of being a <forEach>-Activity"
endNonFunctionalProperties
hasCounterName ofType (1) variable
isParallel ofType (1) boolean
hasStartCounterValue ofType (1) CounterValue
hasFinalCounterValue ofType (1) CounterValue
hasCompletionCondition ofType (0 1) CompletionCondition
hasScope ofType (1) Scope

concept CompletionCondition subConceptOf CounterValue
nonFunctionalProperties
dc:description hasValue "Concept of being a CompletionCondition"
endNonFunctionalProperties
hasSuccessfulBranchesOnly ofType (0 1) boolean

concept CounterValue
nonFunctionalProperties
dc:description hasValue "Concept of being a CounterValue"
endNonFunctionalProperties
hasExpressionLanguage ofType (0 1) _string
hasUnsignedIntegerExpression ofType (1) _string.

concept Scope subConceptOf {StructuredActivity, StandardAttributes}
nonFunctionalProperties
  description hasValue "Concept of being a <scope>-Activity"
endNonFunctionalProperties
isIsolated ofType (0 1) _boolean
hasExitOnStandardFault ofType (0 1) _boolean
hasVariable ofType Variable
hasPartnerLink ofType bpel#PartnerLink
hasMessageExchange ofType MessageExchange
hasCorrelationSet ofType CorrelationSet
hasCatch ofType Catch
catchAll ofType (0 1) CatchAll
hasOnEvent ofType OnEvent
hasOnAlarm ofType RepeatableOnAlarm
hasCompensationHandler ofType (0 1) CompensationHandler
catchHandler ofType (0 1) TerminationHandler
hasActivity ofType (1) Activity
Appendix F  BPM02SBPEL Translator Rules

This appendix contains the ATL (Atlas Transformation Language) rules used to implement the BPM02SBPEL Translator. ATL is part of the Model to Model (M2M) development approach offered by Eclipse (http://wiki.eclipse.org/M2M/ATL). We also list the Java source code used to launch the ATL engine. The translator is available standalone at http://kmi.open.ac.uk/projects/super/BPM02SBPEL-2.0.zip

F.1 Translator Rules

```java
module Bpmo2sBpel; -- Module Template
create OUT : Wsml from IN1 : Wsml, IN2 : syntax;

helper def: bpmoNamespace: String = "http://ip-super.org/ontologies/process/bpmo/v2.0.1#";
helper def: sbpelNamespace: String = "http://ip-super.org/ontologies/process/sbpel/v2.0.0#";
helper def: bpelNamespace: String = "http://ip-super.org/ontologies/process/bpel20/v2.0.0#";

-- this helper (getValueContent) is needed because ATL does not capture content of mixed xml schema types
-- this helper accesses the syntax.ecore metamodel which defines attribute values as strings instead of mixed
-- this is necessary in order to recover the content of attribute values with primitive types
helper def: getValueContent(instName: String, attName: String): String =
let syntaxlnstance: syntax I InstanceType = syntax!InstanceType.allInstances()->any(x| x.name = instName)
in let syntaxAttrValue: syntax!AttributeValueType = syntaxInstance.attributeValue->any(y| y.name = attName)
in syntaxAttrValue.value->first().trim();

helper context WsmllInstanceType def: isInstanceOf(instName: String): Boolean =
self.memberOf->collect(x| x.trim())->includes(instName.trim());

helper context String def: mapNameToSbpel(): String =
self.regexReplaceAll(self.substring(1, self.indexOf('#')), "http://ip-super.org/instances/auto-generated/sBPEL".concat('_sBPEL'), "");

helper context String def: getNameWithNoNamespace(): String =
self.regexReplaceAll(self.substring(1, self.indexOf('#')+1), ".".trim());

helper context WsmllInstanceType def: getProcessWorkflow(): WsmllInstanceType =
let processAttrValue: WsmllAttributeValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasWorkflow')
in let attrValueType: String = processAttrValue.value->first().type
in WsmllInstanceType.allInstances()->any(x| x.name = attrValueType);

helper context WsmllInstanceType def: getWorkflowElement(): WsmllAttributeValueType =
self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasFirstWorkflowElement');

helper context WsmllInstanceType def: getOrganisationAtt(): WsmllAttributeValue =
self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasOrganisation');

helper context WsmllInstanceType def: getParallelSplitBranches(): Sequence(WsmllAnyValue) =
let psAttrValue: WsmllAttributeValue = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasBranch')
in let attrValues: Sequence(WsmllAnyValue) = psAttrValue.value
in attrValues->iterate(e; branches:Sequence(WsmllInstanceType) = Sequence{});
```

299
helper context Wsml!InstanceType def: getBranchElement() : Wsml!AttributeValueType =
  self.attributeValue->any(x | x.name = thisModule.bpmoNamespace+'hasElement');

helper context Wsml!InstanceType def: getDefChoiceBranches() : Sequence(Wsml!InstanceType) =
  let dcAttrValue: WsmllAttributeValueType = self.attributeValue->any(x | x.name = thisModule.bpmoNamespace+
    'hasDefChoice');
in let attrValues: Sequence(Wsml!InstanceType) = Sequence{};
branches.append(Wsml!InstanceType.allInstances()->any(x | x.name = e.type))
}

helper context Wsml!InstanceType def: getBranchEvent(eventName: String) : Wsmll!AttributeValueType =
  self.attributeValue->any(x | x.name = thisModule.bpmoNamespace+'hasEvent');

helper def: getConnectorBySource(sourceName: String) : Wsmll!InstanceType =
  let allConnectors: Sequence(Wsml!InstanceType) =
    Wsml!InstanceType.allInstances()->select(x | x.isInstanceOf(thisModule.bpmoNamespace+'ControlflowConnector'))
in allConnectors->any(y | y.getConnectorSource().value->first().type = sourceName);

helper context Wsml!InstanceType def: getConnectorTarget() : WsmllAttributeValueType =
  self.attributeValue->any(x | x.name = thisModule.bpmoNamespace+'hasTarget');

helper context WsmlllnstanceType def: getConnectorTargetName() : String =
  self.getConnectorTarget().value->first().type;

helper context WsmlllnstanceType def : isConnectorButNotTerminal() : Boolean =
  if self.isInstanceOf(thisModule.bpmoNamespace+'ControlflowConnector')
  then let targetName: String = self.getConnectorTargetName()
in let targetInstance: WsmlllnstanceType = Wsml!lnstanceType.alllnstances()->any(x | x.name = targetName)
in if (targetInstance.isInstanceOf(thisModule.bpmoNamespace+'EndEvent'))
  then false else true endif
  else false endif;

helper def : getEventTypeName(eventName: String) : String =
  let eventInstance : WsmlllnstanceType = Wsml!lnstanceType.alllnstances()->any(x | x.name = eventName)
in if (eventInstance.isInstanceOf(thisModule.bpmoNamespace+'TimerEvent'))
  then 'hasOnAlarm' else 'hasOnMessage'
  endif;

helper context WsmllAttributeValueType def : isElementAttOfReceiveMessageEvent() : Boolean =
  if self.name=thisModule.bpmoNamespace+'hasTarget'
  then if (self.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'ControlflowConnector'))
    then true else false endif
  else false endif;

helper context WsmllAttributeValueType def : isTargetAttOfControlFlowConnector() : Boolean =
  if (self.name=thisModule.bpmoNamespace+'hasTarget')
  then if (self.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'ControlflowConnector'))
    then true else false endif
  else false endif;

helper context WsmllAttributeValueType def : isOutgoingConnectorAttOfParallelSplit() : Boolean =
  if (self.name=thisModule.bpmoNamespace+'hasOutgoingConnector')
  then if (self.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'ControlflowConnector'))
    then true else false endif
  else false endif;

300
helper context Wsm!!InstanceType def: getGoalTaskPartnerGoal() : Wsm!!InstanceType =
let gAttrValue: Wsm!!AttributeValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasPartnerGoal')
in if (not gAttrValue.oclIsUndefined())
then
let attrValue: String = gAttrValue.value->first().type
in Wsm!!InstanceType.allInstances()->any(x| x.name = attrValueType)
else OclUndefined
endif;

helper context Wsm!!InstanceType def: inputDescAttExists() : Boolean =
let idAttrValue: Wsm!!AttributeValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasInputDescription')
in if (idAttrValue.oclIsUndefined())
then false else true endif;

helper context Wsm!!InstanceType def: outputDescAttExists(): Boolean =
let odAttrValue: Wsm!!AttributeValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasOutputDescription')
in if (odAttrValue.oclIsUndefined())
then false else true endif;

helper context Wsm!!InstanceType def: isAtomicTask(): Boolean =
if (self.isInstanceOf(thisModule.bpmoNamespace+'GoalTask')
or self.isInstanceOf(thisModule.bpmoNamespace+'Receive')
or self.isInstanceOf(thisModule.bpmoNamespace+'Send'))
then true else false endif;

helper context Wsm!!InstanceType def: isAtomicTaskWithoutInput(); Boolean =
If (self.isInstanceOf(thisModule.bpmoNamespace+'GoalTask')
or self.isInstanceOf(thisModule.bpmoNamespace+'Receive')
or self.isInstanceOf(thisModule.bpmoNamespace+'DataMediator')
or self.isInstanceOf(thisModule.bpmoNamespace+'ReceiveMessageEvent'))
then if (self.inputDescAttExists() = false)
then true else false endif
else false endif;

helper context Wsm!!InstanceType def: isAtomicTaskWithoutOutputQ : Boolean =
if (self.isInstanceOf(thisModule.bpmoNamespace+'GoalTask')
or self.isInstanceOf(thisModule.bpmoNamespace+'Send')
or self.isInstanceOf(thisModule.bpmoNamespace+'DataMediator'))
then if (self.outputDescAttExistsf) = false)
then true else false endif
else false endif;

helper context Wsm!!InstanceType def: partnerWSAttExists() : Boolean =
let partnerWSAttrValue: Wsm!!AttributeValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasPartnerWebService')
in if (partnerWSAttrValue.oclIsUndefined)
then false else true endif;

helper context Wsm!!InstanceType def: getPartnerWSname() : String =
let wsAttrValue: Wsm!!AttributeValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasPartnerWebService')
in if (not wsAttrValue.oclIsUndefined())
then wsAttrValue.value->first().type else 'noNS#noName' endif;

helper context Wsm!!InstanceType def: getReceiveCounterpartName() : String =
let rcAttrValue: Wsm!!AttributeValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasReceiveCounterpart')
in if (not rcAttrValue.oclIsUndefined())
then rcAttrValue.value->first().type else 'noNS#noName' endif;
helper context Wsml!InstanceType def: isRestrictedSemanticCapability(): Boolean =
  if self.isInstanceOf(thisModule.bpmoNamespace+'SemanticCapability')
  then
    let allRAtt: Sequence(Wsml!AttributeValueValueType) = Wsml!AttributeValueValueType.allInstances()
    ->select(x|x.name = thisModule.bpmoNamespace+'hasPartnerWebService'
      or x.name = thisModule.bpmoNamespace+'hasPartnerGoal'
      or x.name = thisModule.bpmoNamespace+'hasWSdescription'
      or x.name = thisModule.bpmoNamespace+'hasMediator'
      or x.name = thisModule.bpmoNamespace+'hasMediationService'
      or x.name = thisModule.bpmoNamespace+'requestsCapability'
      or x.name = thisModule.bpmoNamespace+'providesCapability')
    in if (allRAtt.isEmpty())
      then true
    else
      let rAttrValue: Wsml!AttributeValueValueType = allRAtt->any(x| x.value->first().type = self.name)
      in if (rAttrValue.oclIsUndefined())
        then true
      else false
    endif
  else false
  endif;

helper context Wsml!InstanceType def: isRestrictedBusinessRole(): Boolean =
  if self.isInstanceOf(thisModule.bpmoNamespace+'BusinessRole')
  then
    let allReceives: Sequence(Wsml!InstanceType) = Wsml!InstanceType.allInstances()
    ->select(x|x.isInstanceOf(thisModule.bpmoNamespace+'Receive'))
    in let selectReceive: Wsml!InstanceType = allReceives->any(z|z.getPartnerRoleName()= self.name)
    in if (selectReceive.oclIsUndefined())
      then false
    else true
    endif
  else false
  endif;

helper def: getReceiveNames(roleName: String): Sequence(String) =
  let allReceives: Sequence(Wsml!InstanceType) = Wsml!InstanceType.allInstances()
  ->select(x|x.isInstanceOf(thisModule.bpmoNamespace+'Receive'))
  in allReceives->select(z|z.getPartnerRoleName()= roleName)
  in allReceives->collect(y|y.name);

helper def: getPartnerRoleTypes(): Sequence(String) =
  let allReceives: Sequence(Wsml!InstanceType) = Wsml!InstanceType.allInstances()
  ->select(x|x.isInstanceOf(thisModule.bpmoNamespace+'Receive'))
  in allReceives->collect(z|z.getPartnerRoleName())->asSet();

helper context Wsml!InstanceType def: getPartnerRoleName(): String =
  let prAttrValue: Wsml!AttributeValueValueType = self.attributeValue->any(x| x.name = thisModule.bpmoNamespace+'hasPartnerRole')
  in if (not prAttrValue.oclIsUndefined())
    then prAttrValue.value->first().type else 'noNS#noName' endif;

helper def: getAllReceives(): Sequence(Wsml!InstanceType) =
  Wsml!InstanceType.allInstances()->select(x|x.isInstanceOf(thisModule.bpmoNamespace+'Receive'));

helper def: getAllInOutNames(): Sequence(String) =
  let allInOut: Sequence(Wsml!AttributeValueValueType) = Wsml!AttributeValueValueType.allInstances()
  ->select(x|x.isInstanceOf(thisModule.bpmoNamespace+'Input'))
  or x.isInstanceOf(thisModule.bpmoNamespace+'Output')
  in allInOut->collect(x|x.value->first().type)->asSet();

helper def: getAllGoalTaskNames(): Sequence(String) =
  let allGoalTasks: Sequence(Wsml!InstanceType) = Wsml!InstanceType.allInstances()
  ->select(x|x.isInstanceOf(thisModule.bpmoNamespace+'GoalTask'))
  in allGoalTasks->collect(x|x.name);
helper def: getOrderedElementForName: String): WsmlInstanceType =
WsmlInstanceType.allInstances().->any(x | x.name = oeName);
-- uses syntax metamodel
helper context WsmlInstanceType def: getOrderedElementOrder(): Integer =
thisModule.getValueContent(self.name, thisModule.bpmoNamespace+'hasOrder').toInt();

-- Called Rules
rule setWsmlAttValueType (attName: String, valueType: String) {
  to
    aAttValue: WsmlAttributeValue (name<-attName, value<-thisModule.setWsmlAttValue(valueType))
  do {aAttValue;}
}
rule setWsmlAttValue (anyValueType: String) {
  to
    aWsmlAnyValue: WsmlWsmlAnyValue (type<-anyValueType)
  do {aWsmlAnyValue;}
}

-- WSML ontology
rule bpmoToSbpel {
  from
    bpmo : Wsml!wsml (bpmo.oclIsTypeOf(Wsml!wsml))
  to
    sbpel : Wsml!wsml (ontology<-on, nonFunctionalProperties<-bpmo.ontology.nonFunctionalProperties, name<-bpmo.ontology.name.mapNameToSbpelO, importsOntology<-Sequence([thisModule.sbpelNamespace+'sbpel', thisModule.bpelNamespace+'bpel20', bpmo.ontology.name.trimO]),
instance<-bpmo.ontology.instance
  ->union(Wsml!InstanceType.allInstances())
    ->collect(y7 | thisModule.resolveTemp(y7, 'aSemanticPick'))
  )
  ->union(Wsml!InstanceType.allInstances())
    ->select(x->false, 'no has ORDER')
    ->collect(y | thisModule.resolveTemp(y, 'aConversation'))
  )
  ->union(Wsml!InstanceType.allInstances())
    ->select(x2 | x2.isAtomicTaskWithoutInput())
303
-- collect(y4| thisModule.resolveTemp(y4, 'anInputSemanticVariable'))

  } ->

-- union(Wsml!InstanceType.allInstances()
  -- select(x5 | x5.isAtomicTaskWithoutOutput())
  -- collect(y5| thisModule.resolveTemp(y5, 'anOutputSemanticVariable'))

  )

-- union(Wsml!AttributeValueType.allInstances()
  -- select(x51 | (x51.name==thisModule.bpmoNamespace+'hasMediator')
  -- or (x51 .name==thisModule.bpmoNamespaceUhasMediationService'))
  -- collect(y51| thisModule.resolveTemp(y51, 'aDataMed'))

  )

-- union(Wsml!InstanceType.allInstances()
  -- select(x6 | x6.isInstanceOf(thisModule.bpmoNamespaceh'TimerEvent')
  -- collect(y6| thisModule.resolveTemp(y6, 'anOnAlarmWaitStatement'))

  )

-- union(Wsml!AttributeValueType.allInstances()
  -- select(x8 | x8.isOutgoingConnectorAttOfParallelSplit())
  -- collect(y8| thisModule.resolveTemp(y8, 'aSeq'))

  )

-- union(Wsml!InstanceType.allInstances()
  -- select(x9 | x9.isRestrictedSemanticCapability())
  -- collect(y9| thisModule.resolveTemp(y9, 'aWSDLMessageType'))

  )

}\)

----- Main concepts
-----

-- points to the workflow element pointed by the hasWorkflow attribute (indirection)

rule processes {
  from i: Wsml!InstanceType ( i.isInstanceOf(thisModule.bpmoNamespace+'Process')
  )
  using {
    goalNames: Sequence(String)= thisModule.getAllGoalTaskNames();
    partnerRoles: Sequence(String)= thisModule.getPartnerRoleTypes();
    --thisModule.getAHReceiveNamesO;
    msgEventNames: Sequence(String)= thisModule.getAllMsgEventNames();
    inOutNames: Sequence(String)= thisModule.getAllInOutNames();
  }
  to
  aProcessInstance: Wsml!InstanceType ( name<-i.name,memberOf<-i.memberOf->select(a|a->trim() o thisModule.bpmoNamespace+'Process')
  --prepend(thisModule.sbpelNamespace+'SemanticProcess'),
  attributeValue<-Sequence{i.getAttributeValue,
    hasmAtt, (If (not partnerRoles.isEmptyO) then hasPartnerAtt else OclUndefined endif),
    (If (not goalNames.isEmptyO) then hasConversationAtt else OclUndefined endif),
    (If (not msgEventNames.isEmptyO) then hasSemOnMesAtt else OclUndefined endif),
    (If (not inOutNames.isEmptyO) then hasVariableAtt else OclUndefined endif),
    correspondsTo Att, i.getProcessWorkflow().getWorkflowElement()}
  )
  hastnAtt: Wsml!AttributeValueType ( name<-thisModule.bpmoNamespace+hasTargetNamespace',
  value<-Sequence{tnValue} )
  xmltype: Wsml!XMLTypeDocumentRoot ( text<-Sequence{'http://ip-super.org/instances/auto-generated/sBPEL'}
  )
  tnValue: Wsml!WsmlAnyValue ( type<-http://www.wsmo.org/wsml/wsml-syntax#string',

304
rule tasks {
  from i: WsmI!InstanceType (
    i.isInstanceOf(thisModule.bpmoNamespace+'Task'))
  to 
    anInstance: WsmI!InstanceType (
      name<-i.name.mapNameToSbpel(),
      memberOf<-i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'Task')
                    ->prepend(thisModule.bpmoNamespace+'Empty'),
      attributeValue<-i.attributeValue
                    ->including(correspondsToAtt)
    ),
  correspondsToAtt: WsmI!AttributeValueType (  
    name<-thisModule.bpelNamespace+'correspondsTo',
    value<-<sequence(hasvalueAtt)
  ),
  hasvalueAtt: WsmI!WsmlAnyValue (  
    type<-i.name
  ),

  hasPartnerAtt: distinct WsmI!AttributeValueType foreach (r in partnerRoles) (  
    name<-thisModule.sbpelNamespace+'hasPartner',
    value<-thisModule.setWsmlAttValue(r.mapNameToSbpel())
  ),

  -- work around xml-wsml serializer
  hasConversationAtt: distinct WsmI!AttributeValueType foreach (c in goalNames) (  
    name<-thisModule.sbpelNamespace+'hasConversation',
    value<-thisModule.setWsmlAttValue(c.mapNameToSbpel().concat('Conversation'))
  ),
  hasVariableAtt: distinct WsmI!AttributeValueType foreach (c in goalNames) (  
    type<-c.mapNameToSbpel().concat('Conversation')
  ),
  hasSemOnMesAtt: distinct WsmI!AttributeValueType foreach (som in msgEventNames)(  
    name<-thisModule.sbpelNamespace+'hasSemanticOnMessage',
    value<-thisModule.setWsmlAttValue(som.mapNameToSbpel())
  ),
  hasVariableAtt: distinct WsmI!AttributeValueType foreach (iod in inOutNames) (  
    name<-thisModule.bpmoNamespace+'hasVariable',
    value<-thisModule.setWsmlAttValue(iod.mapNameToSbpel())
  )
}

rule goalTasks {
  from i: WsmI!InstanceType (  
    i.isInstanceOf(thisModule.bpmoNamespace+'GoalTask'))
  to 
    anExtensionActivity: WsmI!InstanceType (  
      name<-i.name.mapNameToSbpel(),
      memberOf<-i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'GoalTask')
                    ->prepend(thisModule.bpmoNamespace+'Extension Activity'),
      attributeValue<-Sequence {correspondsToAtt, hasActivityAtt}
    ),
  correspondsToAtt: WsmI!AttributeValueType (  
    name<-thisModule.bpelNamespace+'correspondsTo',
    value<-<sequence(hasvalueAtt)
  ),
  hasvalueAtt: WsmI!WsmlAnyValue (  
    type<-i.name
  )
}
null
memberOf:{thisModule.sbpelNamespace+'SemanticVariable'},
attributeValue:<thisModule.setWsmlAttValueType(thisModule.sbpelNamespace+'hasSemanticType',
'http://www.wsmo.org/wsml/wsml-syntax#string')

belongsToConversationAtt: WsmllAttributeValueType(
  name:<thisModule.sbpelNamespace+'belongsToConversation',
  value:Sequence[aValue2])

aValue2: WsmllWsmlAnyValue(
  type:<i.name.mapNameToSbpel().concat('Conversation')>
)

hasNameAtt: WsmllAttributeValueType(
  name:<thisModule.sbpelNamespace+'hasName',
  value:Sequence[aValue])

xmltype: WsmllXMLTypeDocumentRoot(
  text:Sequence<i.name.mapNameToSbpel().concat('Conversation').getNameWithNoNamespace()>)

aValue: WsmllWsmlAnyValue(
  type:'http://www.wsmo.org/wsml/wsml-syntax#string',
mixed:xmltype.mixed
)

hasdescribesInterfaceAtt: WsmllAttributeValueType(
  name:<thisModule.sbpelNamespace+'describesInterface',
  value:Sequence[aValue3])

aValue3: WsmllWsmlAnyValue(
  type:<i.name.mapNameToSbpel().concat('Interface')>
)

anlnterface: WsmlllnstanceType(
  name:<i.name.mapNameToSbpel().concat('Interface'),
  memberOf:{thisModule.sbpelNamespace+'Incominglnterface'},
  attributeValue:Sequence{hasWSDescriptionAtt}
)

hasWSDescriptionAtt: WsmllAttributeValueType(
  name:<thisModule.sbpelNamespace+'hasWebServiceDescription',
  value:Sequence[aValue4])

xmltype1: WsmllXMLTypeDocumentRoot(
  text:<if (i.partnerWSAttExistsO)
  then
  Sequence{i.getPartnerWSname(),thisModule.bpmoNamespace+'hasSemanticDescription'}
  else
  Sequence{""}
  endif>)

aValue4: WsmllWsmlAnyValue(
  type:'http://www.wsmo.org/wsml/wsml-syntax#string',
mixed:xmltype1.mixed
)

rule senders {
  from i: WsmlllnstanceType(
    i.isInstanceOf(thisModule.bpmoNamespace+'Send')
  )
  to
  anExtensionActivity: WsmlllnstanceType(
    name:<i.name.mapNameToSbpel().concat('ExtensionActivity'),
    memberOf:<i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'Send')
    ->prepend(thisModule.bpmoNamespace+'ExtensionActivity'),
    attributeValue:Sequence[correspondsToAtt, hasActivityAtt])
  )
  correspondsToAtt: WsmllAttributeValueType(
    name:<thisModule.bpmoNamespace+'correspondsTo',
    value:Sequence[""],
    namespaceThisModule)
value<-Sequence([hasValueAtt])
)

hasValueAtt: Wsml!WSMLAnyValue(
type<-i.name)
)

hasActivityAtt: Wsml!AttributeValueType(
name<-thisModule.bpemsNamespace+'hasActivity',
value<-Sequence([aValue1])
)

aValue1: Wsml!WSMLAnyValue(
type<-i.name.mapNameToSbpel().concat('Send'))
)

aSendReceive: Wsml!InstanceType(
name<-i.name.mapNameToSbpel().concat('Send'),
memberOf<-Sequence([thisModule.bpemsNamespace+'Send']),
attributeValue<-i.attributeValue->including(belongsToConversationAtt)
attributeValue<-Sequence([i.attributeValue,
belongsToConversationAtt,
(if not i.outputDescAttExists() then hasOutputVarAtt else OclUndefined endif)
)}

hasOutputVarAtt: Wsml!AttributeValueType(
name<-thisModule.bpemsNamespace+'hasVariable',
value<-Sequence([outpVarValue])
)

outpVarValue: Wsml!WSMLAnyValue(
type<-i.name.mapNameToSbpel().concat('SendSemanticVariable'))
)

anOutputSemanticVariable: Wsml!InstanceType(
name<-i.name.mapNameToSbpel().concat('SendSemanticVariable'),
memberOf<-Sequence([thisModule.bpemsNamespace+'SemanticVariable'],
attributeValue<-thisModule.setWsmlAttValueType(thisModule.bpemsNamespace+'hasSemanticType',
'http://www.wsmo.org/wsml/wsml-syntax#string')
)

belongsToConversationAtt: Wsml!AttributeValueType(
name<-thisModule.bpemsNamespace+'belongsToConversation',
value<-Sequence([aValue2])
)

aValue2: Wsml!WSMLAnyValue(
type<-i.getReceiveCounterpartName().mapNameToSbpel().concat('Conversation'))
)

—includes the element part of the same branch as this event

rule receiveMessageEvents {
  from i: Wsml!InstanceType (i.isInstanceOf(thisModule.bpemsNamespace+'ReceiveMessageEvent'))
  to aInstanceOf: Wsml!InstanceType (name<-i.name.mapNameToSbpel(),
    memberOf<-i.memberOf->select(a|a->trim() <> thisModule.bpemsNamespace+'ReceiveMessageEvent')
      ->prepend(thisModule.bpemsNamespace+'SemanticOnMessage'),
    attributeValue<-Sequence([i.attributeValue,
      thisModule.getEventBranch(i.name).getBranchElement(),
    belongsToConversationAtt,
    (if not i.inputDescAttExists() then hasInputVarAtt else OclUndefined endif)
  )
)

hasInputVarAtt: Wsml!AttributeValueType(
name<-thisModule.bpemsNamespace+'hasVariable',
value<-Sequence([inpVarValue])
)

inpVarValue: Wsml!WSMLAnyValue(
type<-i.name.mapNameToSbpel().concat('SemOnMessageSemanticVariable'))
)

anInputSemanticVariable: Wsml!InstanceType (
name<-i.name.mapNameToSbpel().concat('SemOnMessageSemanticVariable'))
)
belongsToConversationAtt: Wsml\AttributeValueType
  
name<-thisModule.sbpelNamespace+'belongsToConversation',
value<-Sequence{aValue2}

aValue2: WsmlAnyValue (
  type<-i.name.mapNameToSbpel().concat('Conversation')
)

aConversation: Wsml\InstanceType (
  name<-i.name.mapNameToSbpel().concat('Conversation')
  ,
  memberOf<->Sequence {thisModule.sbpelNamespace+'Conversation1'}

attributeValue<-Sequence {hasNameAtt, hasdescribesInterfaceAtt}

hasNameAtt: Wsml\AttributeValueType
  
name<-thisModule.sbpelNamespace+'hasName',
value<-Sequence{aValue}

xmltype: Wsml\XMLTypeDocumentRoot (
  text<-Sequence{i.name.mapNameToSbpel().concat('Conversation').getNameWithNoNamespace()}
)

aValue: Wsml\AnyValue (
  type<-http://www.wsmo.org/wsml/wsml-syntax#string,
  mixed<-xmltype.mixed
)

hasdescribesInterfaceAtt: Wsml\AttributeValueType
  
name<-thisModule.sbpelNamespace+'describesInterface',
value<-Sequence{aValue3}

aValue3: Wsml\AnyValue (
  type<-i.name.mapNameToSbpel().concat('IncomingInterface')
)

anInterface: Wsml\InstanceType (
  name<-i.name.mapNameToSbpel().concat('IncomingInterface')
  ,
  memberOf<->Sequence {thisModule.sbpelNamespace+'IncomingInterface'}

attributeValue<-Sequence {hasWSDescriptionAtt}

hasWSDescriptionAtt: Wsml\AttributeValueType
  
name<-thisModule.sbpelNamespace+'hasWebServiceDescription',
value<-Sequence{aValue4}

xmltype1: Wsml\XMLTypeDocumentRoot (
  text<-(if (i.partnerWSAttExists())
    then
      Sequence{thisModule.getValueContent(i.getPartnerWSname(),thisModule.bpmoNamespace+'hasSemanticDescription')}
    else
      Sequence {''}
  endif)
)

aValue4: Wsml\AnyValue (
  type<-http://www.wsmo.org/wsml/wsml-syntax#string',
  mixed<-xmltype1.mixed
)

--includes the element part of the same branch as this event

rule timers {
  
from i: Wsml\InstanceType (i.isInstanceOf(thisModule.bpmoNamespace+'TimerEvent'))

to

anInstance: Wsml\InstanceType (name<-i.name.mapNameToSbpel(),

memberOf<-i.memberOf->select(a->trim() -> thisModule.bpmoNamespace+'TimerEvent')

->prepend(thisModule.bpmoNamespace+'OnAlarm'),

attributeValue<-Sequence {{attributeValue, thisModule.getEventBranch(i.name).getBranchElement(), hasWaitStatementAtt

310
APPENDIX F

```
rule mediationTasks {
  from i: WsmlInstanceType (i.isInstanceOf(thisModule.bpmoNamespace+'MediationTask'))
  to anInstance: WsmlInstanceType (name<-i.name.mapNameToSbpel(),
    memberOf<-i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'MediationTask')
    ->prepend(thisModule.bpmoNamespace+'Assign'),
    attributeValue<-i.attributeValue->including(correspondsToAtt)
  ),
  correspondsToAtt: WsmlAttributeValueType (name<-thisModule.bpmoNamespace+'correspondsTo',
    value<-Sequence{hasvalueAtt}
  ),
  hasvalueAtt: WsmlWsmlAnyValue (type<-i.name)
}

rule hasDataMediatorAtt {
  from aa: WsmlAttributeValueType (aa.name=thisModule.bpmoNamespace+'hasDataMediator')
  to 
  hasWorkElemAtt: distinct WsmlAttributeValueType foreach (med in aa.value)(
    name<-thisModule.bpmoNamespace+'hasAssignOperation',
    value<-aa.value
  )
}

rule dataMediators {
  from i: WsmlInstanceType (i.isInstanceOf(thisModule.bpmoNamespace+'DataMediator'))
  to anInstance: WsmlInstanceType (name<-i.name.mapNameToSbpel(),
    memberOf<-i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'DataMediator')
    ->prepend(thisModule.sbpelNamespace+'Mediate'),
    attributeValue<-$equence {i.attributeValue,
      (if not i.inputDescAttExists() then hasInputVarAtt else OclUndefmed endif),
      (if not i.outputDescAttExists() then hasOutputVarAtt else OclUndefmed endif)
    },
    hasInputVarAtt: WsmlAttributeValueType (name<-thisModule.sbpelNamespace+'hasInputVariable',
      value<-Sequence {inpVarValue}
    ),
    inpVarValue: WsmlWsmlAnyValue (type<-i.name.mapNameToSbpel().concat('MeditatorInputSemanticVariable'))
  ),
  hasOutputVarAtt: WsmlAttributeValueType (name<-i.name.mapNameToSbpel().concat('MeditatorOutputSemanticVariable')
    )
}
```
APPENDIX F

```xml
memberOf<Sequence {thisModule.sbpelNamespace+'SemanticVariable'},
attributeValue=thisModule.setWsmlAttValueType(thisModule.sbpelNamespace+'hasSemanticType',
'http://www.wsmo.org/wsml/wsml-syntax#string')
}).
hasOutputVarAtt: WsmllAttributeValueType {
  name<thisModule.sbpelNamespace+'hasOutputVariable',
  value<Sequence {outVarValue}
}).
outVarValue: WsmllWsmlAnyValue {
  type<i.name.mapNameToSbpel().concat('MediatorOutputSemanticVariable')
}).
anOutputSemanticVariable: WsmlllnstanceType {
  name<i.name.mapNameToSbpel().concat('MediatorOutputSemanticVariable'),
  memberOF<Sequence {thisModule.sbpelNamespace+'SemanticVariable'},
  attributeValue=thisModule.setWsmlAttValueType(thisModule.sbpelNamespace+'hasSemanticType',
'http://www.wsmo.org/wsml/wsml-syntax#string')
}

rule hasMediatorAtt {
  from aa: WsmllAttributeValueType
  (aa.name=thisModule.bpmoNamespace+'hasMediator')
to
  hasWorkElemAtt: WsmllAttributeValueType {
    name<thisModule.sbpelNamespace+'usesDataMediator',
    value<aa.value
  }).
  udmValue: WsmllWsmlAnyValue {
    type<aa.value->first().type.mapNameToSbpel().concat('DataMediator')
  }).
 aDataMed: WsmlllnstanceType {
    name<aa.value->first().type.mapNameToSbpel().concat('DataMediator'),
    memberOF<Sequence {thisModule.sbpelNamespace+'DataMediator'},
    attributeValue<hasMedServAtt
  }).
  hasMedServAtt: WsmllAttributeValueType {
    name<thisModule.sbpelNamespace+'usesMediator',
    value<Sequence {aValue}
  }).
  xmltype: WsmllXMLTypeDocumentRoot {
    text<Sequence {thisModule.getValueContent(aa.value->first().type,thisModule.bpmoNamespace+'hasSemanticDescription')}
  }).
  aValue: WsmllWsmlAnyValue {
    type<http://www.wsmo.org/wsml/wsml-syntax#string',
mixed<xmltype.mixed
  })
}

rule hasMedServiceAtt {
  from aa: WsmllAttributeValueType
  (aa.name=thisModule.bpmoNamespace+'hasMediationService')
to
  hasWorkElemAtt: WsmllAttributeValueType {
    name<thisModule.sbpelNamespace+'usesDataMediator',
    value<Sequence {udmValue}
  }).
  udmValue: WsmllWsmlAnyValue {
    type<aa.value->first().type.mapNameToSbpel().concat('DataMediator')
  }).
 aDataMed: WsmlllnstanceType {
    name<aa.value->first().type.mapNameToSbpel().concat('DataMediator'),
    memberOF<Sequence {thisModule.sbpelNamespace+'DataMediator'},
    attributeValue<hasMedServAtt
  }).
  hasMedServAtt: WsmllAttributeValueType {
    name<thisModule.sbpelNamespace+'usesMediationService',
    value<Sequence {aValue}
  })
}````
rule BusinessRoles {
  from i: Wsm!InstanceType ()
  i.isRestrictedBusinessRole()
  }
  using {
    receiveNames: Sequence(String)= thisModule.getReceiveNames(i.name);
  }
  to aPartner: Wsm!InstanceType ()
  name<-i.name.mapNameToSbpel(),
  memberOf<-i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'BusinessRole')
  ->prepend(thisModule.sbpelNamespace+'Partner'),
  attributeValue<-Sequence({attributeValue,
    thisModule.setOrganisationAtt(i.getOrganisationAtt().value->first(|.type),
    hasConvAtt
  })
  },
  hasConvAtt: distinct Wsm!AttributeValueType foreach (r in receiveNames) {
    name<-thisModule.sbpelNamespace+'hasConversation',
    value<-r.value
  }
  convValue: distinct Wsm!WsmlAnyValue foreach (r in receiveNames) {
    type<-r.mapNameToSbpel()+ 'Conversation'
  }
}

rule setOrganisationAtt(orgInstName: String) {
  to
  hasBEAtt: Wsm!AttributeValueType ()
  name<-thisModule.sbpelNamespace+'hasBusinessEntity',
  value<-Sequence({aValue})
  }
  xmltype: Wsm!XMLTypeDocumentRoot {
    text<-Sequence{thisModule.getValueContent(orgInstName,
      'http://ip-super.Org/ontologies/process/upo/v2.0.1#hasName')}
  }.
  aValue: Wsm!WsmlAnyValue ()
  type<-http://www.wsmo.org/wsml/wsml-syntax#string',
  mixed<-xmltype.mixed
  do {hasBEAtt;}
} -- points to the event elements of the branch (indirection)

rule deferredChoiceMerges {
  from i: Wsm!InstanceType ()
  i.isInstanceOf(thisModule.bpmoNamespace+'DeferredChoiceMerge')
  }
  to
  anExtensionActivity: Wsm!InstanceType ()
  name<-i.name.mapNameToSbpel(),
  memberOf<-i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'DeferredChoiceMerge')
  ->prepend(thisModule.bpmoNamespace+'ExtensionActivity'),
  attributeValue<-Sequence({correspondsToAtt, hasActivityAtt}
  ),
  correspondsToAtt: Wsm!AttributeValueType ()
  name<-thisModule.bpmoNamespace+'correspondsTo',
  value<-Sequence({hasvalueAtt})
  },
hasValueAtt: WsmlAnyValue {
  type<i.name
}
}
hasActivityAtt: WsmlAttributeValueType {
  name<thisModule.bpelNamespace+'hasActivity',
  value<Sequence{aValue1}
}
}
aValue1: WsmlAnyValue {
  type<i.name.mapNameToSbpel().concat('SemanticPick')
}
}
aSemanticPick: WsmlInstanceType {
  name<i.name.mapNameToSbpel().concat('SemanticPick'),
  memberOf<Sequence{thisModule.bpelNamespace+'SemanticPick'},
  attributeValue<i.attributeValue
    --union(i.getDefChoiceBranches()->collect(e|e.getBranchEvent()))
}
}

rule startEvents {
  from i: WsmlInstanceType {
    i.isInstanceOf(thisModule.bpmoNamespace+'StartEvent')
  }
  to anInstance: WsmlInstanceType {
    name<i.name.mapNameToSbpel(),
    memberOf<i.memberOf->select(a|a->trim() <> thisModule.bpmoNamespace+'StartEvent')
      ->prepend(thisModule.bpelNamespace+'Sequence'),
    attributeValue<hasActivityAtt
  },
  hasActivityAtt: WsmlAttributeValueType {
    name<thisModule.bpelNamespace+'hasOrderedActivity',
    value<Sequence{hasValueAtt2}
  },
  hasValueAtt2: WsmlAnyValue {
    type<thisModule.getConnectorBySource(i.name).mapNameToSbpel()
  }
}

rule controlflowConnectors {
  from i: WsmlInstanceType {
    i.isConnectorButNotTerminal()
  }
  to anInstance: WsmlInstanceType {
    name<i.name.mapNameToSbpel(),
    memberOf<i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'ControlflowConnector' 
      ->prepend(thisModule.bpelNamespace+'OrderedActivity'),
    attributeValue<Sequence{i.attributeValue,
      if targetConnector.isConnectorButNotTerminal() 
        then hasNextActivityAtt else OclUndefined endif
    }
  }
  hasNextActivityAtt: WsmlAttributeValueType {
    name<thisModule.bpelNamespace+'hasOrderedActivity',
    value<Sequence{hasValueAtt2}
  },
  hasValueAtt2: WsmlAnyValue {
    type<targetConnector.name.mapNameToSbpel()
  }
}

-- Needs hasOrder and hasNextElement attributes defined in order to map to a list
rule sequences {
  from i: WsmlInstanceType {
    i.isInstanceOf(thisModule.bpmoNamespace+'Sequence')
  }
  to
  314

APPENDIX F
anInstance: Wsml!InstanceType {
    name<-i.name.mapNameToSbpe1(),
    memberOf<-i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'Sequence')
    ->prepend(thisModule.bpmoNamespace+'Sequence'),
    attributeValue<-i.attributeValue
    ->including(correspondsToAtt)
},

hasvalueAtt: Wsml!WsmlAnyValue {
    type<i.name
}

rule orderedElements {
    from i: Wsml!InstanceType {
        i.isInstanceOf(thisModule.bpmoNamespace+'OrderedElement')
    }
    to
    anInstance: Wsml!InstanceType {
        name<-i.name.mapNameToSbpe1(),
        memberOf<-i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'OrderedElement')
        ->prepend(thisModule.bpmoNamespace+'OrderedActivity'),
        attributeValue<-i.attributeValue
        ->including(correspondsToAtt)
    }
    needs hasOrder and hasNextElement attributes defined in order to map to a list

    rule exclusiveChoiceMerges {
        from i: Wsml!InstanceType {
            i.isInstanceOf(thisModule.bpmoNamespace+'ExclusiveChoiceMerge')
        }
        to
        anInstance: Wsml!InstanceType {
            name<-i.name.mapNameToSbpe1(),
            memberOf<-i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'ExclusiveChoiceMerge')
            ->prepend(thisModule.bpmoNamespace+'If'),
            attributeValue<-i.attributeValue
            ->including(correspondsToAtt)
        },
        hasvalueAtt: Wsml!WsmlAnyValue {
            type<i.name
        }
    }

    rule exclusiveChoices {
        from i: Wsml!InstanceType {
            i.isInstanceOf(thisModule.bpmoNamespace+'ExclusiveChoice')
        }
        to
        anInstance: Wsml!InstanceType {
            name<-i.name.mapNameToSbpe1(),
            memberOf<-i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'ExclusiveChoice')
            ->prepend(thisModule.bpmoNamespace+'If'),
            attributeValue<-i.attributeValue
            ->including(correspondsToAtt)
        },
        hasvalueAtt: Wsml!WsmlAnyValue {
            type<i.name
        }
    }
}

315
rule hasDefaultBranchAtt {
  from aa: Wsml! AttributeValue
  (aa.name=thisModule.bpelNamespace+"hasDefaultBranch")
  to
  hasWorkElemAtt: Wsml! AttributeValue
  (name<thisModule.bpelNamespace+"hasElseBranch",
   value<aa.value)
}

rule conditionalBranches {
  from i: Wsml! InstanceType
    (i.isInstanceOf(thisModule.bpmoNamespace+"ConditionalBranch")
  )
  to
  anInstance: Wsml! InstanceType
    (name<i.name.mapNameToSbpel(),
    memberOf<i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+"ConditionalBranch")
    >prepend(thisModule.bpelNamespace+"ConditionalBranch")
    , attributeValue<i.attributeValue
}

rule whiles {
  from i: Wsml! InstanceType
    (i.isInstanceOf(thisModule.bpmoNamespace+"While")
  )
  to
  anInstance: Wsml! InstanceType
    (name<i.name.mapNameToSbpel(),
    memberOf<i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+"While")
    >prepend(thisModule.bpelNamespace+"While")
    , attributeValue<i.attributeValue
}

rule repeats {
  from i: Wsml! InstanceType
    (i.isInstanceOf(thisModule.bpmoNamespace+"Repeat")
  )
  to
  anInstance: Wsml! InstanceType
    (name<i.name.mapNameToSbpel(),
    memberOf<i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+"Repeat")
    >prepend(thisModule.bpelNamespace+"Repeat(_until")
    , attributeValue<i.attributeValue
}

-- points to the branch elements pointed by the hasBranch attribute (indirection)

rule parallelSplitSynchronises {
  from i: Wsml! InstanceType
    (i.isInstanceOf(thisModule.bpmoNamespace+"ParallelSplitSynchronise")
  )
  to
  anInstance: Wsml! InstanceType
    (name<i.name.mapNameToSbpel(),
    memberOf<i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+"ParallelSplitSynchronise")
    >prepend(thisModule.bpelNamespace+"Flow")
    , attributeValue<i.attributeValue
    ->including(correspondsToAtt)
    ->including(hasElementAtt) -- work around xml-wsml serializer
    ->including(i.getParallelSplitBranches()->collect(e|e.getBranchElement()))
  )
  correspondsToAtt: Wsml! AttributeValue
  (name<thisModule.bpelNamespace+"correspondsTo",
   value<Sequence {hasvalueAtt}
rule parallelSplits {
  from i: Wsmll InstanceType (
    i.isInstanceOf(thisModule.bpmoNamespace+'ParallelSplit')
  )
  to anInstance: Wsmll InstanceType ( 
    name<-i.name.mapNameToSbpel(),
    memberOf<-i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'ParallelSplit')
    ->prepend(thisModule.bpmoNamespace+'Flow'),
    attributeValue<-Sequence { i.attributeValue,
      correspondsToAtt
    },
    correspondsToAtt: Wsmll AttributeValueType ( 
      name<-thisModule.bpmoNamespace+'correspondsTo',
      value<-Sequence {hasvalueAtt}
    ),
    hasvalueAtt: Wsmll Wsml AnyValue ( 
      type<-i.name
    )
  )
}

rule conditions {
  from i: Wsmll InstanceType ( 
    i.isInstanceOf(thisModule.bpmoNamespace+'Condition')
  )
  to anInstance: Wsmll InstanceType ( 
    name<-i.name.mapNameToSbpel(),
    memberOf<-i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'Condition')
    ->prepend(thisModule.bpmoNamespace+'Condition'),
    attributeValue<-i.attributeValue
  )
}

rule semanticCapabilities {
  from i: Wsmll InstanceType ( 
    i.isRestrictedSemanticCapability() 
    ->i.isInstanceOf(thisModule.bpmoNamespace+'SemanticCapability')
  )
  to anInstance: Wsmll InstanceType ( 
    name<-i.name.mapNameToSbpel(),
    memberOf<-i.memberOf->select(a|a->trim()) <> thisModule.bpmoNamespace+'SemanticCapability')
    ->prepend(thisModule.bpmoNamespace+'SemanticVariable'),
    attributeValue<-Sequence {hasNameAtt, 
      hasTypeAtt, 
      i.attributeValue
    },
    hasNameAtt: Wsmll AttributeValueType ( 
      name<-thisModule.bpmoNamespace+'hasName',
      value<-Sequence {aValue}
    ),
    xmltype: Wsmll XMLTypeDocumentRoot ( 
      text<-Sequence {i.name.mapNameToSbpel().getNameWithNoNamespace()} 
    ),
    aValue: Wsmll Wsml AnyValue ( 
      type<-'http://www.wsmo.org/wsml/wsml-syntax#string',
      mixed<-'xmltype.mixed'
    ),
    hasTypeAtt: Wsmll AttributeValueType ( 
      name<-thisModule.bpmoNamespace+'hasType',
      value<-Sequence {hasTypeValue}
    ),
    hasTypeValue: Wsmll Wsml AnyValue ( 
      type<-'http://www.wsmo.org/wsml/wsml-syntax#string',
      mixed<-'xmltype.mixed'
    )
  )

317
type<i>.name.mapNameToSbpel().concat('WSDL.MessageType')
}

aWSDL.MessageType: WsmllInstanceType (
  name<i>.name.mapNameToSbpel().concat('WSDL.MessageType'),
  memberOf<-Sequence {thisModule.bpelNamespace+'WSDL.MessageType'},
  attributeValue<-thisModule.setWsmlAttValueType(thisModule.bpelNamespace+'hasDefinition',
    'http://www.wsmo.org/wsmo/wsml-syntax#string')
})

rule hasSemanticDescriptionAtt {
  from aa: WsmllAttributeValueType
  (aa.name=thisModule.bpmoNamespace+'hasSemanticDescription')
  using {
    parentName: String = aa.refImmediateComposite().name;
  }
  to
  hasAtt1: WsmllAttributeValueType {
    name<-thisModule.bpmoNamespace+'hasSemanticType',
    value<-Sequence {aValue}
  },
  xmltype: WsmllXMLTypeDocumentRoot {
    text<-Sequence {thisModule.getValueContent(parentName, aa.name)}
  },
  aValue: WsmllWsmlAnyValue {
    type<-aa.value->first().type,
    mixed<-xmltype.mixed
  }
}

rule subprocesses {
  from i: WsmllInstanceType {
    i.isInstanceOf(thisModule.bpmoNamespace+'SubProcess')
  }
  to
  anInstance: WsmllInstanceType {
    name<i>.name.mapNameToSbpel(),
    memberOf<-Sequence {a|a->trim() <> thisModule.bpmoNamespace+'SubProcess'},
    attributeValue<i.attributeValue
    ->including(correspondsToAtt)
  },
  correspondsToAtt: WsmllAttributeValueType {
    name<-thisModule.bpmoNamespace+'correspondsTo',
    value<-Sequence {hasvalueAtt}
  },
  hasvalueAtt: WsmllWsmlAnyValue {
    type<i.name
  }
}

----- Attributes

----- rule hasConditionAtt {
  from aa: WsmllAttributeValueType
  (aa.name=thisModule.bpmoNamespace+'hasCondition')
  to
  hasWorkElemAtt: WsmllAttributeValueType {
    name<-thisModule.bpmoNamespace+'hasCondition',
    value<-aa.value
  }
}

rule hasExpressionAtt {
  from aa: WsmllAttributeValueType
  (aa.name=thisModule.bpmoNamespace+'hasExpression')
  using {
    parentName: String = aa.refImmediateComposite().name;
  }

318
APPENDIX F

```
to hasAtt: Wsmll/AttributeValueType {
    name<-thisModule.bpelNamespace+'hasExpression',
    value<-Sequence(aa.value)
},
xmltype: Wsmll/XMLTypeDocumentRoot {
    text<-Sequence(thisModule.getValueContent(parentName,aa.name))
},
aValue: Wsmll/AnyValue {
    type<-aa.value->first().type,
    mixed<-xmltype.mixed
}
}

rule hasWorkflowElementAtt {
    from aa: Wsmll/AttributeValueType {
        name<-thisModule.bpmoNamespace+'hasFirstWorkflowElement'
    } to
    hasWorkElemAtt: Wsmll/AttributeValueType {
        name<-thisModule.bpelNamespace+'hasActivity',
        value<-aa.value
    }
}

rule executesAtt {
    from aa: Wsmll/AttributeValueType {
        name<-thisModule.bpmoNamespace+'executes'
    } to
    hasWorkElemAtt: Wsmll/AttributeValueType {
        name<-thisModule.bpelNamespace+'hasActivity',
        value<-aa.value
    }
}

-- points to first element (order = 1) only
rule hasOrderedElementAtt {
    from aa: Wsmll/AttributeValueType {
        name<-thisModule.bpmoNamespace+'hasOrderedElement'
    } to
    hasOEAtt: Wsmll/AttributeValueType {
        name<-thisModule.bpelNamespace+'hasOrderedActivity',
        value<-aa.value->any(e|thisModule.getOrderedElement(e.type).getOrderedElementOrder(|=1))
    }
}

rule hasElementAtt {
    from aa: Wsmll/AttributeValueType {
        name<-thisModule.bpmoNamespace+'hasElement'
    } to
    hasAtt: Wsmll/AttributeValueType {
        name<-if (aa.isElementAttOfReceiveMessageEvent()) then thisModule.bpmelNamespace+'hasActivity' else thisModule.bpmoNamespace+'hasActivity' endif,
        value<-aa.value
    }
}

rule hasConditionalBranchAtt {
    from aa: Wsmll/AttributeValueType {
        name<-thisModule.bpmoNamespace+'hasConditionalBranch'
    } to
    hasAtt: Wsmll/AttributeValueType {
        name<-thisModule.bpmoNamespace+'hasIfBranch'
        value<-aa.value->any(e|thisModule.getOrderedElement(e.type).getOrderedElementOrder(|=1))
    }
}

rule hasOutgoingConnectorAtt {
```

319
from aa: Wsmu! AttributeValueType
( aa.isOutgoingConnectorAttOfParallelSplit() )
to
  aWorkElemAtt: distinct Wsmu! AttributeValueType foreach ( connector in aa.value)
    name<-thisModule.bpmoNamespace+'hasActivity',
    value<-thisModule.setWsmuAttValue(connector.type.mapNameToSbpel().concat('Sequence'))
},
  aSeq: distinct Wsmu! InstanceType foreach ( connector in aa.value)
    name<-connector.type.mapNameToSbpel().concat('Sequence'),
    memberOf<-thisModule.bpmoNamespace+'Sequence',
    attributeValue<-hasOrdActAtt
},
  hasOrdActAtt: distinct Wsmu! AttributeValueType foreach ( connector in aa.value) ( name<-thisModule.bpmoNamespace+'hasOrderedActivity',
    value<-ordActValue
),
  ordActValue: distinct Wsmu! WsmuAnyValue foreach ( connector in aa.value)
    type<-connector.type.mapNameToSbpel()
}

rule hasInputDescriptionAtt {
  from aa: Wsmu! AttributeValueType
  ( aa.name=thisModule.bpmoNamespace+'hasInputDescription' )
to
  hasWorkElemAtt: Wsmu! AttributeValueType ( name<-thisModule.sbpelNamespace.concat( if ( aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'GoalTask')
    or aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'DataMediator')
    then 'hasInputVariable' else 'hasVariable' endif ),
    value<-aa.value
  )
}

rule hasOutputDescriptionAtt {
  from aa: Wsmu! AttributeValueType
  ( aa.name=thisModule.bpmoNamespace+'hasOutputDescription' )
to
  hasWorkElemAtt: Wsmu! AttributeValueType ( name<-thisModule.sbpelNamespace.concat( if ( aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'GoalTask')
    or aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'DataMediator')
    then 'hasOutputVariable' else 'hasVariable' endif ),
    value<-aa.value
  )
}

-- checks the type of event to give it a name
rule hasEventAtt {
  from aa: Wsmu! AttributeValueType
  ( aa.name=thisModule.bpmoNamespace+'hasEvent' )
to
  hasAtt: Wsmu! AttributeValueType ( name<-thisModule.sbpelNamespace.concat(thisModule.getEventTypeName(aa.value->first().type)),
    value<-aa.value
  )
}

rule hasNextElementAtt {
  from aa: Wsmu! AttributeValueType
  ( aa.name=thisModule.bpmoNamespace+'hasNextElement' )
to
  hasAtt: Wsmu! AttributeValueType ( name<-thisModule.bpmoNamespace.concat( if ( aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'OrderedElement')
    then 'hasOrderedActivity' else 'hasElseBranch' endif ),
    value<-aa.value
  )
}
rule hasTargetAtt {
  from aa: Wsmll! AttributeValueType
    (aa.isTargetAttOfControlFlowConnector())
  to
    hasAtt: Wsmll! AttributeValueType (
      name<-thisModule.bpmoNamespace+'hasActivity';
      value<-aa.value
    )
}

-- uses syntax metamodel to get name values
rule hasNameAtt {
  from aa: Wsmll! AttributeValueType
    (aa.name=thisModule.bpmoNamespace+'hasName')
  using {
    parentName: String = aa.refImmediateComposite().name;
  }
  to
    hasAtt1: Wsmll! AttributeValueType (
      name<-(if (aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'Process'))
       or aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'MediationTask')
       or aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'TimerEvent')
       or aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'SubProcess')
       or aa.refImmediateComposite().isInstanceOf(thisModule.bpmoNamespace+'Task')
      then thisModule.bpmoNamespace+'hasName'
      else thisModule.sbpelNamespace+'hasName' endif),
      value<Sequence{aValue}
    ),
    xmltype: Wsmll! XMLTypeDocumentRoot (text<Sequence{'Automatically generated by BPMO2sBPEL Tool'}),
    aValue: Wsmll! WsmllAnyValue (
      type<-aa.value->first().type,
      mixed<-xmltype.mixed
    )
}

rule hasCreatorAtt {
  from aa: Wsmll! AttributeValueType
    (aa.name='http://purl.org/dc/elements/1.1#creator')
  to
    hasAtt2: Wsmll! AttributeValueType (
      name<-aa.name,
      value<Sequence{aValue}
    ),
    xmltype: Wsmll! XMLTypeDocumentRoot (text<Sequence{'Automatically generated by BPMO2sBPEL Tool'}),
    aValue: Wsmll! WsmllAnyValue (
      type<-aa.value->first().type,
      mixed<-xmltype.mixed
    )
}

rule nips {
  from i: Wsmll! NonFunctionalPropertiesType
  to
    nfp: Wsmll! NonFunctionalPropertiesType (attributeValue<-i.attributeValue)
}

rule attValues {
  from i: Wsmll! WsmllAnyValue
  to
    aValue: Wsmll! WsmllAnyValue (}
A ppendix F
type<-i.type.mapNameToSbpel( )

)

}

F.2 Translator Java Source
package kmi.open.ac.uk;

/**
* BPM02sBPEL Translator

*

* SUPER EU IP Project
* @author Liliana Cabral
* @version 2.0
* @created October, 2008
*/
import java.io.*;
import java.util. Map;
import java.util.HashMap;
import java.net. URL;
import java.net.MalformedURLException;
import java.util.Collections;
import org.apache.commons. logging. Log;
import org.apache.commons.logging.LogFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import j avax .xml .transform. OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactoty;
import org.wsmo.common.*;
import org.wsmo.factory.*;
import org.wsmo.wsml.*;
import org.eclipse.m2m.atl.drivers.emf4atl.ASMEMFModel;
import org.eclipse.m2m.atl.engine. AtlEMFModelHandler;
import org.eclipse.m2m.atl.engine.AtlLauncher;
import org.eclipse.m2m.atl.engine.AtlModelHandler;

/**
* BPM02sBPEL Translator.
* SUPER EU IP Project.

*

* @author Liliana Cabral
* @version 2.0.
* Created October 2008.

*/

public class BPM02sBPEL {
private static final Log logger = LogFactory.getLog(BPM02sBPEL.class);
private final static String resourcePath = "./resources/”;
private TransformerFactory transfomerFactory = null;
private WsmoFactory factory = null;

322


private LogicalExpressionFactory leFactory;
private DataFactory dataFactory = null;
private Parser parser;
private Serializer xmlSerializer;
private Serializer wsmlSerializer;
private Parser xmlParser;
private File myBPMOfiile;
private File tempATLXMLFile;
private File tempATLXMLSyntaxFile;
private File tempATLXMISyntaxFile;
private File tempATLXMIOutfile;
private AtlEMFModelHandler modelHandler;
private File WSML_ModelResource;
private File WSMLWithoutMixedValues_ModelResource;
private URL BPM02SBPEL_TransfoResource;
private ASMEMFModel wsmlMetamodel;
private ASMEMFModel syntaxMetamodel;

/**
 * Translates a BPMO instance (BPMOv2.0.1) to a sBPEL instance.
 * @param bpmoFileName. Full path name of a Wsml file containing a BPMO Instance
 * @param sbpelFileName. Full path Wsml file name of the generated sBPEL
 */
public static void translate(String bpmoFileName, String sbpelFileName) throws IOException, ParserException, Exception {
    BPM02sBPEL translator = new BPM02sBPEL();
    translator.parseBPMO(bpmoFileName,sbpelFileName);
    translator.atlTransformation();
    logger.info("Finished Successfully.");
}

private void parseBPMO(String bpmoFileName, String sbpelFileName) {
    FileReader wsmlReader = null;
    TopEntity[] entities = null;
    File tempXMLFile = null;
    try {
        myBPMOfile = new File(bpmoFileName);
        mySBPELfile = new File(sbpelFileName);
        if (myBPMOfile.exists) return;
        if (! myBPMOfile.getName().endsWith(".wsml") || ! mySBPELfile.getName().endsWith(".wsmx"))
            throw new RuntimeException("Input/Output file name must have .wsml extension");
        mySBPELfile.createNewFile();
        logger.info("input file: " + myBPMOfile.toString());
        FlashMap <String, Object> props = new HashMap<String, Object>();
        // use default implementation for factory
        factory = Factory.createWsmoFactory(null);
        leFactory = Factory.createLogicalExpressionFactory(null);
        dataFactory = Factory.createDataFactory(null);
        props.put(Factory.WSMO_FACTORY, factory);
        props.put(Factory.LE_FACTORY, leFactory);
        props.put(Factory.DATA_FACTORY, dataFactory);
        parser = Factory.createServer(props);
        translator = Factory.createServer(props);
        logger.info("input file: " + myBPMOfile.toString());
        HashMap <String, Object> propsXMLSer = new HashMap<String, Object>();
        propsXMLSer.put(org.wsmo.factory.Factory.PROVIDERCLASS, com.ontotext.wsmo4j.serializer.xml.WsmlXmlSerializer.class.getName());
        xmlSerializer = Factory.createServer(propsXMLSer);
        propsXMLSer.put(org.wsmo.factory.Factory.PROVIDERCLASS, com.ontotext.wsmo4j.parser.xml.WsmlXmlParser.class.getName());
        xmlParser = Factory.createServer(propsXMLSer);
    }
}
propsXMLParser.put(Factory.DATA_FACTORY, dataFactory);

xmlParser = Factory.createParser(propsXMLParser);

HashMap<String, Object> propsWSMLSer = new FIashMap<String, Object>();
propsWSMLSer.put(org.wsmo.factory.Factory.PROVIDERCLASS,
com.ontotext.wsmo4j.serializer.wsml.WSMLSerializerImpl.class.getNameO);

wsmlSerializer = org.wsmo.factory.Factory.createSerializer(propsWSMLSer);

wsmlReader=new FileReader(myBPMOfile);
entities = parser.parse(new BufferedReaderi wsml Reader));
tempXMLFile = File.createTempFile(myBPMOfile.getName()+"xml", ".xml", null);
tempATLXMIfile = File.createTempFile(myBPMOfile.getName()+"xmi", ".xmi", null);
tempATLXMISyntaxFile = File.createTempFile(myBPMOfile.getName()+"xmiSyntax", ".xmi", null);

FileWriter xmlWriter = new FileWriter(tempXMLFile);
xmlSerializer.serializelentities, new BufferedWriter(xmlWriter));

transfomerFactory = TransformerFactory.newInstanceO;
Transformer transformer = transfomerFactory ,newTransformer();
transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARAT10N, "no");
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty(OutputKeys.ENCODING, "ISO-8859-1");

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
// parse the input file to get a Document object
Document doc = db.parse(tempXMLFile);
Element WsmlElement = doc.getDocumentElement();

//creating xmi wsml metamodel
Element XMIElement = doc.createElement("xmi:XMI");
XMIElement.setAttribute("xmi:version", "2.0");
XMIElement.setAttribute("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
XMIElement.setAttribute("xmlns:xm1", "Wsml");

XMIElement.appendChild(WsmlElement);

DOMSource source1 = new DOMSource(XMIElement);
transformer.transform(source1, new StreamResult(tempATLXMlfile));

//creating xmi syntax metamodel
Element XMISyntaxElement = doc.createElement("xml:XMI");
XMISyntaxElement.setAttribute("xmi:version", "2.0");
XMISyntaxElement.setAttribute("xmlns:xm1", "http://www.w3.org/2001/XMLSchema-instance");
XMISyntaxElement.setAttribute("xmlns:xm1", "syntax");
XMISyntaxElement.appendChild(WsmlElement);

DOMSource source2 = new DOMSource(XMISyntaxElement);
transformer.transform(source2, new StreamResult(tempATLXMISyntaxFile));

} catch (IOException ex) {
ex.printStackTrace();
}

} catch (ParserException ex) {
System.out.println("Invalid WSML token encountered at line " + ex.getErrorLine() + ", position " + ex.getErrorPos());
}

} catch (Exception ex) {
ex.printStackTrace();
}

}

private void atlTransformation( ) {

//logger.debug("Started atlTransformation. ");
try {


tempATLXMIOutfile = File.createTempFile(myBPMOfile.getName() + "xmiOut", ".xmi", null);
bpmo2sbpel(tempATLXMIfile, tempATLXMISyntaxFile, tempATLXMIOutfile);

FileReader xmlReader = new FileReader(tempATLXMlOutfile);
TopEntity[] xmlTopEntities = xmlParser.parse(new BufferedReader(xmlReader));

if (xmlTopEntities.length != 1)
    throw new RuntimeException("The number of parsed entities is not 1!");

FileWriter wsmlWriter = new FileWriter(mySBPELfile);
wsmlSerializer.serialize(xmlTopEntities, wsmlWriter);
logger.info("generated file: " + mySBPELfile.toString());
}

} catch (IOException ex) {
ex.printStackTrace();
}
}
}
}
}

private void createResources() throws MalformedURLException {
    modelHandler = (AtlEMFModelHandler) AtlModelHandler.getDefault(AtlModelHandler.AMHEMF);

    File BPM02SBPEL_ATL_File = new File(resourcePath + "BPM02sBPEL.asm");
    BPM02SBPEL_TransfoResource = BPM02SBPEL_ATL_File.toURI().toURL();

    
    private void initMetamodels(Map<String, Object> models) throws IOException {
        WSML ModelResource = new File(resourcePath + "Wsm1.ecore");
        WSML_WithoutMixedValues_ModelResource = new File(resourcePath + "syntax.ecore");
        wsmlMetamodel = (ASMEMFModel) modelHandler.loadModel("Wsm1", modelHandler.getMof(),
                new FileInputStream(WSMLModelResource));
        syntaxMetamodel = (ASMEMFModel) modelHandler.loadModel("syntax", modelHandler.getMof(),
                new FileInputStream(WSML_WithoutMixedValues_ModelResource));
        models.put("Wsm1", wsmlMetamodel);
        models.put("syntax", syntaxMetamodel);
        //logger.debug("Finished initMetamodels. ");
    
    private void bpmo2sbpel(File inFilePath, File in2FilePath, File outFilePath) {
        try {
            Map<String, Object> models = new HashMap<String, Object>();
            createResources();
            initMetamodels(models);
            //get/create models
            ASMEMFModel bpmoInputModel = (ASMEMFModel) modelHandler.loadModel("IN1", wsmlMetamodel,
                    new FileInputStream(inFilePath));
            ASMEMFModel bpmoSyntaxInputModel = (ASMEMFModel) modelHandler.loadModel("IN2", syntaxMetamodel,
                    new FileInputStream(in2FilePath));
            ASMEMFModel sbpelOutputModel = (ASMEMFModel) modelHandler.newModel("OUT",
                    outFilePath.toURI().toASCIIString(), wsmlMetamodel);
            //load models
            models.put("IN1", bpmoInputModel);
            models.put("IN2", bpmoSyntaxInputModel);
            models.put("OUT", sbpelOutputModel);
            //launch
            AllLauncher.getDefault().launch(this.BPM02SBPEL_TransfoResource,
                    Collections.EMPTY_MAP, models, Collections.EMPTY_MAP,
                    Collections.EMPTY_LIST, Collections.EMPTY_MAP);
        }
modelHandler.saveModel(sbpelOutputModel, new FileOutputStream(tempATLXMIOutfile));

private void dispose(Map<String, Object> models) {
    for (Object model : models.values()) {
        ((ASMEMModel) model).dispose();
    }
}

/**
 * Usage: BPM02sBPEL
 * -o=outputWSMLFileName inputBPMOWsmlFileName
 */

public static void main(String[] args) throws IOException, ParserException, Exception {
    String outputFile = null;
    String inputFile = null;

    logger.info("BPM02sBPEL Translator");
    if (args.length < 2 || args.length > 2) {
        logger.info("Usage: BPM02sBPEL " +
                "-o=outputWSMLFileName <inputBPMOWsmlFileName>*");
        System.exit(1);
    }

    for (int i = 0; i < args.length; i++)
        if (args[i].startsWith("-o")) {
            String[] tokens = args[i].split("=");
            outputFile = tokens[1];
        } else
            inputFile = args[i];

    BPM02sBPEL.translate(inputFile, outputFile);
}
Appendix G  Semantic Descriptions used in the Use Case on E-government

This appendix contains the semantic descriptions used in the Change of Circumstances scenario presented in the use case on e-government as well as the WSDL for some of the services implemented. The implementation of the prototype can be found at http://kmi.open.ac.uk/projects/dip/DIP-wp9-ChangeOfCircumstances.zip

G.1 Semantic Descriptions for the Change of Circumstances Scenario

The following ontology is under ocml\library\v5-0\goals\change-of-circumstances-prototype

;;; Mode: Lisp; Package: ocml
;;; File created in WebOnto
(in-package "OCML")
(in-ontology change-of-circumstances-prototype)

(DEF-CLASS GET-CITIZEN-DATA-GOAL
  (GOAL)
  (?GOAL
   ((HAS-INPUT-ROLE :VALUE FAMILY_NAME :VALUE FIRST_NAMES)
    (HAS-INPUT-SOAP-BINDING :VALUE
      (FAMILY_NAME "string")
      (FIRST_NAMES "string")
    )
    (HAS-OUTPUT-ROLE :VALUE CITIZEN-DATA RECORDS)
    (HAS-OUTPUT-SOAP-BINDING :VALUE
      (CITIZEN_DATA_RECORDS "xml")
      (FAMILY_NAME :TYPE STRING)
      (FIRST_NAMES :TYPE STRING)
      (CITIZEN_DATA_RECORDS :TYPE CITIZEN-DATA) )))

(DEF-CLASS GET-CITIZEN-DATA-WEB-SERVICE
  (WEB-SERVICE)
  (?WEB-SERVICE
   ((HAS-INPUT-ROLE :VALUE FAMILY_NAME :VALUE FIRST_NAMES)
(HAS-OUTPUT-ROLE :VALUE CITIZEN_DATA_RECORDS)
(FAMILY-NAME :TYPE STRING)
(FIRST-NAME :TYPE STRING)
(CITIZEN_DATA_RECORDS :TYPE CITIZEN-DATA)
(HAS-CAPABILITY :VALUE GET-CITIZEN-DATA-CAPABILITY)
(HAS-INTERFACE :VALUE GET-CITIZEN-DATA-INTERFACE)
(HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CITIZEN-DATA-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS GET-CITIZEN-DATA-CAPABILITY
  (CAPABILITY)
  (?CAPABILITY
   ((USED-MEDIATOR :VALUE GET-CITIZEN-DATA-MEDIATOR)
    (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CITIZEN-DATA-CAPABILITY-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS GET-CITIZEN-DATA-INTERFACE
  (INTERFACE)
  (?INTERFACE
   ((HAS-CHOREOGRAPHY :VALUE GET-CITIZEN-DATA-INTERFACE-CHOREOGRAPHY)
    (HAS-ORCHESTRATION :VALUE GET-CITIZEN-DATA-INTERFACE-ORCHESTRATION)
    (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CITIZEN-DATA-INTERFACE-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS get-citizen-data-orchestration
  (ORCHESTRATION)

(DEF-CLASS GET-CITIZEN-DATA-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
  (PROBLEM-SOLVING-PATTERN)
  NIL)

(DEF-CLASS GET-CITIZEN-DATA-MEDIATOR
  (WG-MEDIATOR)
  (?MEDIATOR
   ((HAS-SOURCE-COMPONENT :VALUE GET-CITIZEN-DATA-GOAL))

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-MEDIATOR
  (WG-MEDIATOR)
  (?MEDIATOR
   ((HAS-SOURCE-COMPONENT :VALUE GET-CITIZEN-DATA-BY-CODE-GOAL))

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-WEB-SERVICE
  (WEB-SERVICE)
  (?WEB-SERVICE
   ((HAS-INPUT-ROLE :VALUE CITIZEN_KEY)
    (HAS-OUTPUT-ROLE :VALUE CITIZEN_DATA_RECORDS)
    (CITIZEN_KEY :TYPE INT)
    (CITIZEN_DATA_RECORDS :TYPE CITIZEN-DATA)
    (HAS-CAPABILITY :VALUE GET-CITIZEN-DATA-BY-CODE-CAPABILITY)
    (HAS-INTERFACE :VALUE GET-CITIZEN-DATA-BY-CODE-INTERFACE)
    (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CITIZEN-DATA-BY-CODE-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-CAPABILITY
  (CAPABILITY)

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-MEDIATOR (INTERFACE) 
(INTERFACE) 
(HAS-CHOREOGRAPHY :VALUE GET-CITIZEN-DATA-BY-CODE-MEDIATOR-CHOREOGRAPHY) 
(HAS-ORCHESTRATION :VALUE GET-CITIZEN-DATA-BY-CODE-MEDIATOR-ORCHESTRATION) 
(HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CITIZEN-DATA-BY-CODE-MEDIATOR-NON-FUNCTIONAL-PROPERTIES))) 

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-INTERFACE (INTERFACE) 
(INTERFACE) 
(HAS-CHOREOGRAPHY :VALUE GET-CITIZEN-DATA-BY-CODE-INTERFACE-CHOREOGRAPHY) 
(HAS-ORCHESTRATION :VALUE GET-CITIZEN-DATA-BY-CODE-INTERFACE-ORCHESTRATION) 
(HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CITIZEN-DATA-BY-CODE-INTERFACE-NON-FUNCTIONAL-PROPERTIES))) 

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-ORCHESTRATION (ORCHESTRATION) 

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN (PROBLEM-SOLVING-PATTERN) NIL) 

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-GOAL (GOAL) 
(GOAL) 
(HAS-INPUT-ROLE :VALUE CITIZEN_KEY) 
(HAS-INPUT-SOAP-BINDING :VALUE (CITIZEN_KEY "int")) 
(HAS-OUTPUT-ROLE :VALUE CITIZEN_DATA_RECORDS) 
(HAS-OUTPUT-SOAP-BINDING :VALUE (CITIZEN_DATA_RECORDS "xml")) 
(CITIZEN_KEY :TYPE INT) 
(CITIZEN_DATA_RECORDS :TYPE CITIZEN-DATA))) 

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-GOAL (GOAL) 
(GOAL) 
(HAS-INPUT-ROLE :VALUE CITIZEN_KEY) 
(HAS-INPUT-SOAP-BINDING :VALUE (CITIZEN_KEY "int")) 
(HAS-OUTPUT-ROLE :VALUE CITIZEN_ADDRESS_RECORDS) 
(HAS-OUTPUT-SOAP-BINDING :VALUE (CITIZEN_ADDRESS_RECORDS "xml")) 
(CITIZEN_KEY :TYPE INT) 
(CITIZEN_ADDRESS_RECORDS :TYPE CITIZEN-ADDRESS))) 

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-MEDIATOR (MG-MEDIATOR) 
(MEDIATOR) 
(HAS-SOURCE-COMPONENT :VALUE GET-CITIZEN-ADDRESS-BY-CODE-GOAL)) 

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-WEB-SERVICE (WEB-SERVICE) 
(WEB-SERVICE) 
(HAS-INPUT-ROLE :VALUE CITIZEN_KEY) 
(HAS-OUTPUT-ROLE :VALUE CITIZEN_DATA_RECORDS) 
(CITIZEN_KEY :TYPE INT) 
(CITIZEN_DATA_RECORDS :TYPE CITIZEN-DATA) 
(HAS-CAPABILITY 329)
APPENDIX G

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-CAPABILITY
  (CAPABILITY)
  (USED-MEDIATOR :VALUE GET-CITIZEN-ADDRESS-BY-CODE-MEDIATOR)
  (HAS-NON-FUNCTIONAL-PROPERTIES
   :VALUE GET-CITIZEN-ADDRESS-BY-CODE-CAPABILITY-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-INTERFACE
  (INTERFACE)
  (HAS-CHOREOGRAPHY :VALUE GET-CITIZEN-ADDRESS-BY-CODE-INTERFACE-CHOREOGRAPHY)
  (HAS-ORCHESTRATION :VALUE GET-CITIZEN-ADDRESS-BY-CODE-INTERFACE-ORCHESTRATION)
  (HAS-NON-FUNCTIONAL-PROPERTIES
   :VALUE GET-CITIZEN-ADDRESS-BY-CODE-INTERFACE-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS get-citizen-address-by-code-orchestration
  (ORCHESTRATION)
  (HAS-PROBLEM-SOLVING-PATTERN :VALUE get-citizen-address-by-code-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN)))

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
  (PROBLEM-SOLVING-PATTERN) NIL)

(DEF-CLASS CHANGE-CITIZEN-DETAILS-MEDIATOR
  (WG-MEDIATOR)
  (MEDIATOR
   (HAS-SOURCE-COMPONENT :VALUE CHANGE-CITIZEN-DETAILS-GOAL)))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-WEB-SERVICE
  (WEB-SERVICE)
  (HAS-INPUT-ROLE
   :VALUE ADDRESS_KEY
   :VALUE POST_CODE
   :VALUE PREMISE_NUMBER
   :VALUE PREMISE_NAME
   :VALUE STREET
   :VALUE LOCALITY
   :VALUE CITIZEN_TOWN)
  (HAS-OUTPUT-ROLE :VALUE CHANGEDETAILS_ACK)
  (ADDRESS_KEY :TYPE INT)
  (POST_CODE :TYPE STRING)
  (PREMISE_NUMBER :TYPE STRING)
  (PREMISE_NAME :TYPE STRING)
  (STREET :TYPE STRING)

330
(LOCALITY :TYPE STRING)
(CITIZEN_TOWN :TYPE STRING)

(CHANGEDETAILS_ACK :VALUE CHANGE-CITIZEN-DETAILS-ABILITY)
(CHANGE-CITIZEN-DETAILS-INTERFACE)

HAS-NON-FUNCTIONAL-PROPERTIES

:VALUE

CHANGE-CITIZEN-DETAILS-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN))

(DEF-CLASS change-citizen-details-orchestration
(ORCHESTRATION)
((HAS-PROBLEM-SOLVING-PATTERN
:VALUE

change-citizen-details-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
(PROBLEM-SOLVING-PATTERN)
NIL)

(DEF-CLASS CHANGE-CITIZEN-DETAILS-GOAL
(GOAL)

?GOAL

((HAS-INPUT-ROLE

:VALUE

ADDRESS_KEY

:VALUE POST_CODE

:VALUE

PREMISE_NUMBER

:VALUE

PREMISE_NAME

:VALUE STREET

:VALUE LOCALITY

:VALUE CITIZEN_TOWN)

HAS-INPUT-SOAP-BINDING

:VALUE

(ADDRESS_KEY "int")

:VALUE

POST_CODE "string")

:VALUE

(PREMISE_NUMBER "string")

:VALUE

(PREMISE_NAME "string")

:VALUE

(STREET "string")

:VALUE

(LOCALITY "string")

:VALUE

(CITIZEN_TOWN "string")

(HAS-OUTPUT-ROLE :VALUE CHANGEDETAILS_ACK)

(HAS-OUTPUT-SOAP-BINDING :VALUE CHANGEDETAILS_ACK "xml")

(ADDRESS_KEY :TYPE INT)

(POST_CODE :TYPE STRING)

(PREMISE_NUMBER :TYPE STRING)

(PREMISE_NAME :TYPE STRING)

(STREET :TYPE STRING)

(LOCALITY :TYPE STRING)
(CITIZEN_TOWN :TYPE STRING)
(CHANGEDETAILS_ACK :TYPE STRING)
(USED-MEDIATOR :VALUE CHANGE-CITIZEN-DETAILS-GOAL-GG-MEDIATOR))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-MEDIATOR
  (WG-MEDIATOR)
  (?MEDIANOR
   ((HAS-SOURCECOMPONENT :VALUE CHANGE-CITIZEN-DETAILS-GOAL))))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-INTERFACE
  (INTERFACE)
  (?INTERFACE
   ((HAS-CHOREOGRAPHY
     :VALUE
     CHANGE-CITIZEN-DETAILS-INTERFACE-CHOREOGRAPHY)
    (HAS-ORCHESTRATION
     :VALUE
     CHANGE-CITIZEN-DETAILS-INTERFACE-ORCHESTRATION)
    (HAS-NONFUNCTIONAL-PROPERTIES
     :VALUE
     CHANGE-CITIZEN-DETAILS-INTERFACE-NONFUNCTIONAL-PROPERTIES)))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-GOAL-GG—MEDIATOR-GG-MEDIATOR
  (HAS-SOURCECOMPONENT :VALUE CHANGE-ADDRESS-GOAL)
  (HAS-TARGETCOMPONENT :VALUE CHANGE-CITIZEN-DETAILS-GOAL)
  (HAS-MEDIATIONSERVICE :VALUE CHANGE-CITIZEN-DETAILS-GOAL-GG—MEDIATOR-MEDIATION-SERVICE)))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-GOAL-GG-MEDIATOR-MEDIATION-SERVICE (GOAL) ?GOAL
  ((HASINPUTROLE
    :VALUE POST_CODE
    :VALUE PREMISE_NUMBER
    :VALUE PREMISE_NAME
    :VALUE STREET
    :VALUE CITIZEN_TOWN
  (HASINPUTSOAPBINDING
    :VALUE (POST_CODE "string")
    :VALUE (PREMISE_NUMBER "string")
    :VALUE (PREMISE_NAME "string")
    :VALUE (STREET "string")
    :VALUE (CITIZEN_TOWN "string")
  (HASOUTPUTROLE
    :VALUE ADDRESSDETAILS
  (HASOUTPUTSOAPBINDING
    :VALUE (ADDRESSDETAILS "xml")
  (POST_CODE :TYPE STRING)
  (PREMISE_NUMBER :TYPE STRING)
  (PREMISE_NAME :TYPE STRING)
  (STREET :TYPE STRING)
  (CITIZEN_TOWN :TYPE STRING)
  (ADDRESSDETAILS :TYPE CITIZENADDRESS)))

(DEF-CLASS CREATE-CITIZEN-GOAL
  (GOAL)
  (?GOAL
  ((HASINPUTROLE
    :VALUE FAMILY_NAME
    :VALUE FIRST_NAMES
    :VALUE INITIALS
    :VALUE GENDER_CODE
    :VALUE MARITAL_STATUS_CODE
    :VALUE TITLE_CODE
    :VALUE ETHNICITY_CODE

332
APPENDIX G

(VALUE SPEECH IMPAIRMENT

(VALUE HEARING IMPAIRMENT

(VALUE DATE OF BIRTH

(VALUE AGE

(VALUE POST_CODE

(VALUE PREMISE_NUMBER

(VALUE PREMISE_NAME

(VALUE STREET

(VALUE LOCALITY

(VALUE CITIZEN_TOWN

(VALUE ADDRESS_TYPE_CODE

(VALUE START_DATE)

(HAS-INPUT-SOAP-BINDING

:VALUE

(FAMILY_NAME "string")

:VALUE

(FIRST_NAMES "string")

:VALUE

(INITIALS "string")

:VALUE

(GENDER_CODE "int")

:VALUE

(MARITAL_STATUS_CODE "int")

:VALUE

(TITLE_CODE "int")

:VALUE

(ETHNICITY_CODE "int")

:VALUE

(SPEECH_IMPAIRMENT "string")

:VALUE

(HEARING_IMPAIRMENT "string")

:VALUE

(DATE_OF_BIRTH "string")

:VALUE

(AGE "string")

:VALUE

(POST_CODE "string")

:VALUE

(PREMISE_NUMBER "string")

:VALUE

(PREMISE_NAME "string")

:VALUE

(STREET "string")

:VALUE

(LOCALITY "string")

:VALUE

(CITIZEN_TOWN "string")

:VALUE

(ADDRESS_TYPE_CODE "int")

:VALUE

(START_DATE "string")

(HAS-OUTPUT-ROLE :VALUE CREATE_CITIZEN_ACK)

(HAS-OUTPUT-SOAP-BINDING

:VALUE

(CREATE_CITIZEN_ACK "string")

(FAMILY_NAME :TYPE STRING)

(FIRST_NAMES :TYPE STRING)

(INITIALS :TYPE STRING)

(GENDER_CODE :TYPE INT)

(MARITAL_STATUS_CODE :TYPE INT)

(TITLE_CODE :TYPE INT)

(ETHNICITY_CODE :TYPE INT)

(SPEECH_IMPAIRMENT :TYPE STRING)

(HEARING_IMPAIRMENT :TYPE STRING)

(DATE_OF_BIRTH :TYPE STRING)

(AGE :TYPE STRING)

(POST_CODE :TYPE STRING)

(PREMISE_NUMBER :TYPE STRING)

(PREMISE_NAME :TYPE STRING)

(STREET :TYPE STRING)
(DEF-CLASS CREATE-CITIZEN-MEDIATOR
  (WG-MEDIATOR)
  (?MEDIATOR)
  ((HAS-SOURCE-COMPONENT :VALUE CREATE-CITIZEN-GOAL)))

(DEF-CLASS CREATE-CITIZEN-WEB-SERVICE
  (WEB-SERVICE)
  (?WEB-SERVICE)
  (HAS-INPUT-ROLE
    :VALUE FAMILY_NAME
    :VALUE FIRST_NAMES
    :VALUE INITIALS
    :VALUE GENDER_CODE
    :VALUE MARITAL_STATUS_CODE
    :VALUE TITLE_CODE
    :VALUE ETHNICITY_CODE
    :VALUE SPEECH_IMPAIEMENT
    :VALUE HEARING_IMPAIEMENT
    :VALUE DATE_OF_BIRTH
    :VALUE AGE
    :VALUE POST_CODE
    :VALUE PREMISE_NUMBER
    :VALUE PREMISE_NAME
    :VALUE STREET
    :VALUE LOCALITY
    :VALUE CITIZEN_TOWN
    :VALUE ADDRESS_TYPE_CODE
    :VALUE START_DATE)
  (HAS-OUTPUT-ROLE :VALUE CREATE-CITIZEN_ACK)
  (FAMILY_NAME :TYPE STRING)
  (FIRST_NAMES :TYPE STRING)
  (INITIALS :TYPE STRING)
  (GENDER_CODE :TYPE INT)
  (MARITAL_STATUS_CODE :TYPE INT)
  (TITLE_CODE :TYPE INT)
  (ETHNICITY_CODE :TYPE INT)
  (SPEECH_IMPAIEMENT :TYPE STRING)
  (HEARING_IMPAIEMENT :TYPE STRING)
  (DATE_OF_BIRTH :TYPE STRING)
  (AGE :TYPE STRING)
  (POST_CODE :TYPE STRING)
  (PREMISE_NUMBER :TYPE STRING)
  (PREMISE_NAME :TYPE STRING)
  (STREET :TYPE STRING)
  (LOCALITY :TYPE STRING)
  (CITIZEN_TOWN :TYPE STRING)
  (ADDRESS_TYPE_CODE :TYPE INT)
  (START_DATE :TYPE STRING)
  (CREATE_CITIZEN_ACK :TYPE STRING))

(DEF-CLASS CREATE-CITIZEN-CAPABILITY
  (CAPABILITY)
  (?CAPABILITY)
  ((USED-MEDIATOR :VALUE CREATE-CITIZEN-MEDIATOR)
    (HAS-NON-FUNCTIONAL-PROPERTIES
     :VALUE CREATE-CITIZEN-CAPABILITY-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS CREATE-CITIZEN-INTERFACE
  (?INTERFACE)
  ((USED-MEDIATOR :VALUE CREATE-CITIZEN-MEDIATOR)
    (HAS-NON-FUNCTIONAL-PROPERTIES
     :VALUE CREATE-CITIZEN-INTERFACE-NON-FUNCTIONAL-PROPERTIES)))

APPENDIX G
(INTERFACE)
  (?INTERFACE
   ((HAS-CHOREOGRAPHY
     VALUE
     CREATE-CITIZEN-INTERFACE-CHOREOGRAPHY)
    (HAS-ORCHESTRATION
     VALUE
     CREATE-CITIZEN-INTERFACE-ORCHESTRATION)
    (HAS-NON-FUNCTIONAL-PROPERTIES
     VALUE
     CREATE-CITIZEN-INTERFACE-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS create-citizen-orchestration
  (ORCHESTRATION)
  ((HAS-PROBLEM-SOLVING-PATTERN
    VALUE
    create-citizen-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN)))

(DEF-CLASS CREATE-CITIZEN-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
  (PROBLEM-SOLVING-PATTERN
   NIL))

(DEF-CLASS REDIRECT-EQUIPMENT-GOAL
  (GOAL)
  (?GOAL
   ((HAS-INPUT-ROLE
      VALUE
      CITIZEN_KEY
      VALUE
      NEW_ADDRESS
      VALUE
      NEW_PHONE_NUMBER)
    (HAS-INPUT-SOAP-BINDING
     VALUE
     (CITIZEN_KEY "string")
     VALUE
     (NEW_ADDRESS "string")
     VALUE
     (NEW_PHONE_NUMBER "string")
    (HAS-OUTPUT-ROLE :VALUE REDIRECT-EQUIPMENT_ACK)
    (HAS-OUTPUT-SOAP-BINDING
     VALUE
     (REDIRECT-EQUIPMENT_ACK "xml"))
    (CITIZEN_KEY :TYPE STRING)
    (NEW_ADDRESS :TYPE STRING)
    (NEW_PHONE_NUMBER :TYPE STRING)
    (REDIRECT-EQUIPMENT_ACK :TYPE STRING)
    (USED-MEDIATOR :VALUE REDIRECT-EQUIPMENT-GOAL-OO-MEDIATOR)
    (HAS-NON-FUNCTIONAL-PROPERTIES
     VALUE
     REDIRECT-EQUIPMENT-GOAL-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS REDIRECT-EQUIPMENT-GOAL-OO-MEDIATOR
  (OO-MEDIATOR)
  (?MEDIATOR
   ((HAS-SOURCE-COMPONENT :VALUE CHANGE-OF-CIRCUMSTANCES-PROTOTYPE)
    (HAS-TARGET-COMPONENT :VALUE CHANGE-OF-CIRCUMSTANCES-PROTOTYPE)
    (HAS-MAPPING-RULES
     VALUE
     (redirect-address-mapping
      (NEW_ADDRESS "REDIRECT-EQUIPMENT-GOAL ?na)
      IF
      (CHANGE-ADDRESS-GOAL ?CHANGE-ADDRESS-GOAL)
      (PREMISE_NUMBER ?CHANGE-ADDRESS-GOAL ?pn)
      (STREET ?CHANGE-ADDRESS-GOAL ?s)
      (CITIZEN_TOWN ?CHANGE-ADDRESS-GOAL ?t)
      (POST_CODE ?CHANGE-ADDRESS-GOAL ?pc)
      (= ?na (string-append ?pn " " ?s " " ?t " " ?pc))))
    (HAS-NON-FUNCTIONAL-PROPERTIES
     VALUE
     REDIRECT-EQUIPMENT-GOAL-NON-FUNCTIONAL-PROPERTIES)))

335
(DEF-CLASS REDIRECT-EQUIPMENT-MEDIATOR
  (WG-MEDIATOR)
  (HAS-SOURCE-COMPONENT :VALUE REDIRECT-EQUIPMENT-GOAL))

(DEF-CLASS REDIRECT-EQUIPMENT-WEB-SERVICE
  (WEB-SERVICE)
  (HAS-INPUT-ROLE :VALUE CITIZEN_KEY :VALUE NEW_ADDRESS :VALUE NEW_PHONE_NUMBER)
  (HAS-OUTPUT-ROLE :VALUE REDIRECT-EQUIPMENT_ACK)
  (HAS-CAPABILITY :VALUE REDIRECT-EQUIPMENT-CAPABILITY)
  (HAS-INTERFACE :VALUE REDIRECT-EQUIPMENT-INTERFACE)
  (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE REDIRECT-EQUIPMENT-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS REDIRECT-EQUIPMENT-CAPABILITY
  (CAPABILITY)
  (USED-MEDIATOR :VALUE REDIRECT-EQUIPMENT-MEDIATOR)
  (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE REDIRECT-EQUIPMENT-CAPABILITY-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS REDIRECT-EQUIPMENT-INTERFACE
  (INTERFACE)
  (HAS-CHOREOGRAPHY :VALUE REDIRECT-EQUIPMENT-INTERFACE-CHOREOGRAPHY)
  (HAS-ORCHESTRATION :VALUE REDIRECT-EQUIPMENT-INTERFACE-ORCHESTRATION)
  (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE REDIRECT-EQUIPMENT-INTERFACE-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS REDIRECT-EQUIPMENT-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
  (PROBLEM-SOLVING-PATTERN)
  NIL)

(DEF-CLASS CHANGE-ADDRESS-GOAL
  (GOAL)
  (HAS-INPUT-SOAP-BINDING :VALUE (ADDRESS_KEY "int") :VALUE (POST_CODE "string"))
(DEF-CLASS CHANGE-ADDRESS-MEDIATOR
  (WG-MEDIATOR)
  (?MEDIATOR
   (HAS-SOURCE-COMPONENT :VALUE CHANGE-ADDRESS-GOAL)))

(DEF-CLASS CHANGE-ADDRESS-WEB-SERVICE
  (WEB-SERVICE)
  (?WEB-SERVICE
   (HAS-INPUT-ROLE
    :VALUE ADDRESSKEY
    :VALUE POST_CODE
    :VALUE PREMISE_NUMBER
    :VALUE PREMISE_NAME
    :VALUE STREET
    :VALUE LOCALITY
    :VALUE CITIZEN_TOWN
    :VALUE CITIZEN_KEY)
   (HAS-OUTPUT-ROLE :VALUE CHANGE_ADDRESS_ACK)
   (ADDRESSKEY :TYPE INT)
   (POST_CODE :TYPE STRING)
   (PREMISE_NUMBER :TYPE STRING)
   (PREMISE_NAME :TYPE STRING)
   (STREET :TYPE STRING)
   (LOCALITY :TYPE STRING)
   (CITIZEN_TOWN :TYPE STRING)
   (CITIZEN_KEY :TYPE INT)
   (CHANGE_ADDRESS_ACK :TYPE STRING))

(DEF-CLASS CHANGE-ADDRESS-CAPABILITY
  (CAPABILITY)
  (?CAPABILITY
   (USED-MEDIATOR :VALUE CHANGE-ADDRESS-MEDIATOR)
   (HAS-NON-FUNCTIONAL-PROPERTIES
    :VALUE CHANGE-ADDRESS-CAPABILITY-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS CHANGE-ADDRESS-INTERFACE
  (INTERFACE)
  (?INTERFACE
   (HAS-CHOREOGRAPHY
    :VALUE CHANGE-ADDRESS-INTERFACE-CHOREOGRAPHY)
   (HAS-ORCHESTRATION
    :VALUE CHANGE-ADDRESS-INTERFACE-ORCHESTRATION)))
CHANGE-ADDRESS-INTERFACE-ORCHESTRATION
(HAS-NON-FUNCTIONAL-PROPERTIES
  VALUE
  CHANGE-ADDRESS-INTERFACE-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS CHANGE-ADDRESS-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
  (PROBLEM-SOLVING-PATTERN)
  ((HAS-BODY
    VALUE
    (ORCH-SEQ
     CHANGE-CITIZEN-DETAILS-GOAL
     REDIRECT-EQUIPMENT-GOAL)
     (RETURN (ORCH-GET-GOAL-VALUE REDIRECT-EQUIPMENT-GOAL))))))

(DEF-CLASS GET-CITIZEN-DATA-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES)
  NIL)

(DEF-CLASS GET-CITIZEN-DATA-CAPABILITY-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES)
  NIL)

(DEF-CLASS GET-CITIZEN-DATA-INTERFACE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES)
  NIL)

(DEF-CLASS GET-CITIZEN-DATA-INTERFACE-CHOREOGRAPHY
  (CHOREOGRAPHY)
  ((HAS-GROUNDING
    VALUE
    (GROUNDED-TO-JAVA
     (NORMAL
      ("com.sap.research.dip.wp9.CitizenDataByName.client.CitizenDataByNameTestClient"  
        "getCitizenData"))))))

(DEF-CLASS GET-CITIZEN-DATA-INTERFACE-ORCHESTRATION
  (ORCHESTRATION)
  ((HAS-PROBLEM-SOLVING-PATTERN
    VALUE
    GET-CITIZEN-DATA-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN))

(DEF-CLASS GET-CITIZEN-DATA-INTERFACE-INTERFACEPUBLISHER-INFORMATION
  (PUBLISHER-INFORMATION)
  ((HAS-ASSOCIATED-WEB-SERVICE-INTERFACE
    VALUE
    GET-CITIZEN-DATA-INTERFACE)
   (HAS-WEB-SERVICE-HOST :VALUE "137.108.25.117")
   (HAS-WEB-SERVICE-PORT :VALUE 8080)
   (HAS-WEB-SERVICE-LOCATION :VALUE "/IRSPublisher/JavaClassInvoker"))

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES)
  NIL)

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-CAPABILITY-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES)
  NIL)

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-INTERFACE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES)
  NIL)

(DEF-CLASS GET-CITIZEN-ADDRESS-BY-CODE-INTERFACE-CHOREOGRAPHY
  (CHOREOGRAPHY)
  ((HAS-GROUNDING
    VALUE
    (GROUNDED-TO-JAVA
     (NORMAL
      ("com.sap.research.dip.wp9.CitizenDataByName.client.CitizenDataByNameTestClient"  
        "getAddressByCode"))))))
APPENDIX G

(VALUE (((GROUNDED-TO-JAVA (NORMAL ("com.sap.research.dip.wp9.CitizenAddressByCode.client.CitizenAddressByCodeClient" "getCitizenAddress"))))))


(DEF-CLASS CHANGE-CITIZEN-DETAILS-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS CHANGE-CITIZEN-DETAILS-CAPABILITY-NON-FUNCTIONAL-PROPERTIES (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS CHANGE-CITIZEN-DETAILS-INTERFACE-NON-FUNCTIONAL-PROPERTIES (NON-FUNCTIONAL-PROPERTIES) NIL)


(DEF-CLASS CHANGE-CITIZEN-DETAILS-INTERFACE-ORCHESTRATION (ORCHESTRATION) ((HAS-PROBLEM-SOLVING-PATTERN :VALUE CHANGE-CITIZEN-DETAILS-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN))))

(DEF-CLASS REDIRECT-EQUIPMENT-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS REDIRECT-EQUIPMENT-CAPABILITY-NON-FUNCTIONAL-PROPERTIES (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS REDIRECT-EQUIPMENT-INTERFACE-NON-FUNCTIONAL-PROPERTIES (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS REDIRECT-EQUIPMENT-INTERFACE-ORCHESTRATION
  (ORCHESTRATION)
  ((HAS-PROBLEM-SOLVING-PATTERN :VALUE
    REDIRECT-EQUIPMENT-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN)))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-WEB-SERVICE-PUBLISHER-INFORMATION
  (PUBLISHER-INFORMATION)
  ((HAS-ASSOCIATED-WEB-SERVICE-INTERFACE :VALUE
    CHANGE-CITIZEN-DETAILS-INTERFACE)
   (HAS-WEB-SERVICE-HOST :VALUE "137.108.25.117")
   (HAS-WEB-SERVICE-PORT :VALUE 8080)
   (HAS-WEB-SERVICE-LOCATION :VALUE
    "/IRSPublisher/JavaClassInvoker"))

(DEF-CLASS REDIRECT-EQUIPMENT-WEB-SERVICE-PUBLISHER-INFORMATION
  (PUBLISHER-INFORMATION)
  ((HAS-ASSOCIATED-WEB-SERVICE-INTERFACE :VALUE
    REDIRECT-EQUIPMENT-INTERFACE)
   (HAS-WEB-SERVICE-HOST :VALUE "137.108.25.117")
   (HAS-WEB-SERVICE-PORT :VALUE 8080)
   (HAS-WEB-SERVICE-LOCATION :VALUE
    "/IRSPublisher/JavaClassInvoker"))

(DEF-CLASS CHANGE-ADDRESS-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS CHANGE-ADDRESS-CAPABILITY-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS CHANGE-ADDRESS-INTERFACE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS CHANGE-ADDRESS-INTERFACE-CHOREOGRAPHY (CHOREOGRAPHY) NIL)

(DEF-CLASS CHANGE-ADDRESS-INTERFACE-ORCHESTRATION
  (ORCHESTRATION)
  ((HAS-PROBLEM-SOLVING-PATTERN :VALUE
    CHANGE-ADDRESS-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN)))

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-WEB-SERVICE-PUBLISHER-INFORMATION
  (PUBLISHER-INFORMATION)
  ((HAS-ASSOCIATED-WEB-SERVICE-INTERFACE :VALUE
    GET-CITIZEN-DATA-BY-CODE-INTERFACE)
   (HAS-WEB-SERVICE-HOST :VALUE "137.108.25.117")
   (HAS-WEB-SERVICE-PORT :VALUE 8080)
   (HAS-WEB-SERVICE-LOCATION :VALUE
    "/IRSPublisher/JavaClassInvoker"))

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-CAPABILITY-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-INTERFACE-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES) NIL)

340
(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-INTERFACE-CHOREOGRAPHY
(CHOREOGRAPHY)
(HAS-GROUNDING :VALUE
(GROUNDED-TO-JAVA NORMAL

(DEF-CLASS GET-CITIZEN-DATA-BY-CODE-INTERFACE-ORCHESTRATION
(ORCHESTRATION)
(HAS-PROBLEM-SOLVING-PATTERN :VALUE
GET-CITIZEN-DATA-BY-CODE-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN))

(DEF-CLASS CREATE-CITIZEN-WEB-SERVICE-PUBLISHER-INFORMATION
(PUBLISHER-INFORMATION)
(HAS-ASSOCIATED-WEB-SERVICE-INTERFACE :VALUE
CREATE-CITIZEN-INTERFACE)
(HAS-WEB-SERVICE-PORT :VALUE "8080")
(HAS-WEB-SERVICE-LOCATION :VALUE
"/IRSPublisher/JavaClassInvoker")

(DEF-CLASS REDIRECT-EQUIPMENT-GOAL-NON-FUNCTIONAL-PROPERTIES
(NON-FUNCTIONAL-PROPERTIES)
NIL)

(DEF-CLASS GET-CATALOGUE-DATA-GOAL-NON-FUNCTIONAL-PROPERTIES
(NON-FUNCTIONAL-PROPERTIES)
NIL)

(DEF-CLASS GET-CATALOGUE-DATA-GOAL
(GOAL)
?GOAL
(HAS-INPUT-ROLE :VALUE HAS_MAX_CLIENT_WEIGHT)
(HAS-INPUT-SOAP-BINDING :VALUE
341)
(HAS_MAX_CLIENT_WEIGHT "int")
(HAS_OUTPUT-ROLE :VALUE CATALOGUE_DATA_ENTRIES)
(HAS_OUTPUT-SOAP-BINDING
 :VALUE
 (CATALOGUE_DATA_ENTRIES "xml")
(HAS_MAX_CLIENT_WEIGHT :TYPE INTEGER)
(CATALOGUE_DATA_ENTRIES :TYPE CATALOGUE-DATA)
(HAS_NON-FUNCTIONAL-PROPERTIES
 :VALUE
 GET-CATALOGUE-DATA-GOAL-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES
 (NON-FUNCTIONAL-PROPERTIES)
 NIL)

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE
 (WEB-SERVICE)
 ?WEB-SERVICE
 (HAS-CAPABILITY
 :VALUE
 GET-CATALOGUE-DATA-WEB-SERVICE-CAPABILITY)
 (HAS-INTERFACE
 :VALUE
 GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE)
 (HAS_NON-FUNCTIONAL-PROPERTIES
 :VALUE
 GET-CATALOGUE-DATA-WEB-SERVICE-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-CAPABILITY-NON-FUNCTIONAL-PROPERTIES
 (NON-FUNCTIONAL-PROPERTIES)
 NIL)

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-CAPABILITY
 (CAPABILITY)
 ?CAPABILITY
 (USED-MEDIATOR :VALUE GET-CATALOGUE-DATA-MEDIATOR)
 (HAS_NON-FUNCTIONAL-PROPERTIES
 :VALUE
 GET-CATALOGUE-DATA-WEB-SERVICE-CAPABILITY-NON-FUNCTIONAL-PROPERTIES))

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-NON-FUNCTIONAL-PROPERTIES
 (NON-FUNCTIONAL-PROPERTIES)
 NIL)

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-CHOREOGRAPHY
 (CHOREOGRAPHY)
 (HAS_GROUNDING
 :VALUE
 (GROUNDED-TO-JAVA
 NORMAL
 ("com.sap.research.dip.wp9.catalogueEntry.client.CatalogueEntryByWeightClient"
 "getCatalogueData"))))

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
 (PROBLEM-SOLVING-PATTERN)
 NIL)

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-ORCHESTRATION
 (ORCHESTRATION)
 (HAS_PROBLEM-SOLVING-PATTERN
 :VALUE
 GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-ORCHESTRATION-PROBLEM-SOLVING-PATTERN))

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE
 (INTERFACE)
 ?INTERFACE
 (HAS_CHOREOGRAPHY
 :VALUE
 GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-CHOREOGRAPHY)
(HAS-ORCHESTRATION :VALUE GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-ORCHESTRATION)
(HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS GET-CATALOGUE-DATA-MEDIATOR-NON-FUNCTIONAL-PROPERTIES
  (NON-FUNCTIONAL-PROPERTIES) NIL)

(DEF-CLASS GET-CATALOGUE-DATA-MEDIATOR
  (WG-MEDIATOR)
  (?MEDIATOR
   (HAS-SOURCE-COMPONENT :VALUE GET-CATALOGUE-DATA-GOAL)
   (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE GET-CATALOGUE-DATA-MEDIATOR-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS GET-CATALOGUE-DATA-WEB-SERVICE-PUBLISHER-INFORMATION
  (PUBLISHER-INFORMATION)
  (HAS-ASSOCIATED-WEB-SERVICE-INTERFACE :VALUE GET-CATALOGUE-DATA-WEB-SERVICE-INTERFACE)
  (HAS-WEB-SERVICE-HOST :VALUE "137.108.25.117")
  (HAS-WEB-SERVICE-PORT :VALUE 8080)
  (HAS-WEB-SERVICE-LOCATION :VALUE "/IRSPublisher/JavaClassInvoker"))

G.2 Domain Ontologies for the Change of Circumstances

Scenario

The following ontologies were under ocm\library\v5-0\domains\swift-services-ontology and under ocm\library\v5-0\domains\elms-services-ontology
(has-HearingImpairment :type string) (has-FirstNames :type string) (has-Initials :type string) (has-DateOfBirth :type date) (has-DateOfDeath :type date) (has-ApproxDateOfBirth :type date) (has-Age :type integer) (has-ExpectedDateOfBirth :type date))

(in-ontology elms-services-ontology)

(def-class catalogue-data ()
  ((has-product-code hype string) (has-description :type string) (has-cost :type string) (has-max-user-weight :type integer) (has-charging-value :type string) (has-product-widtht :type string) (has-product-height :type string) (has-product-seat-height :type string) (has-product-depth :type string) (has-technician-fit hype boolean) (has-product-weight hype string) (has-narrative-detail hype string) (has-essex-ss-ot-dept :type string) (has-main-supplier hype string) (has-telephone-number :type string) (has-fax-number :type string) (has-category :type string))

G.3 WSDL for the Change of Circumstances Scenario

Below we provide the WSDL files of three Web Services used in examples in Chapter 5: CatalogueEntryByWeightInterfaceOut, ChangeDetailsOfCitizenInterfaceOut; and RedirectEquipmentToNewAddressInterfaceOut.

```
<xml version="1.0" encoding="ISO-8859-1">
name="CatalogueEntryByWeightInterfaceOut" targetNamespace="http://sap.com/research/dip/wp9/elmdb">
  <wSDL:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://sap.com/research/dip/wp9/elmdb"
targetNamespace="http://sap.com/research/dip/wp9/elmdb">
      <xsd:element name="CatalogueEntryResponseMessage" type="CatalogueEntryResponseType" />
      <xsd:element name="CatalogueEntryByWeightRequestMessage" type="CatalogueEntryByWeightRequestType" />
      <xsd:complexType name="CatalogueEntryByWeightRequestType">
        <xsd:annotation>
          <xsd:appinfo source="http://sap.com/xi/TextID">8f3b0ea1d72e1d9b033d2f59b3831cc</xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element name="MaxClientWeight" type="xsd:integer" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wSDL:types>
</wSDL:definitions>
```
<xsd:appinfo source="http://sap.com/xi/TextID">
  8f3b0eaf2d72e11d9b437d2f99b3f831ec
</xsd:appinfo>
</xsd:element>
</xsd:complexType>
</xsd:complexType name="CatalogueEntryResponseType">
<xsd:annotation>
  <xsd:appinfo source="http://sap.com/xi/TextID">
    3f580c70d7211d9a746000c29518eaf
  </xsd:appinfo>
</xsd:annotation>
</xsd:sequence>
<xsd:complexType>
<xsd:complexContent>
  <xsd:sequence>
    <xsd:element name="CatalogueData" minOccurs="0" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:appinfo source="http://sap.com/xi/TextID">
          3665bbb0d72c11d9b325d2f99b3f831ec
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="ProductCode" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb1d72c11d9b8a24d2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="Description" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb2d72c11d9a378d2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="Cost" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb3d72c11d9b5ded2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="MaxUserWeight" type="xsd:integer">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb4d72c11d995cbd2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="ChargingValue" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb5d72c11d985bd0d2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="ProductWidth" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb6d72c11d9b2a4d2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="ProductHight" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb7d72c11d9a67ad2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="ProductCategory" type="xsd:string">
            <xsd:annotation>
              <xsd:appinfo source="http://sap.com/xi/TextID">
                3665bbb8d72c11d99324d2f99b3f831ec
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="CatalogueEntryByWeightRequestMessage" type="p1:CatalogueEntryByWeightRequestType" />
</xsd:message>
</xsd:operation>
</xsd:port>
</xsd:message>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
</wsdl:types>
<wsdl:message name="CatalogueEntryByWeightInterfaceOut">
<wsdl:part name="CatalogueEntryByWeightInterfaceOut'' element=''p1 :CatalogueEntryByWeightRequestMessage'' />
</wsdl:message>
<wsdl:message name="CatalogueEntryResponseMessage">
<wsdl:part name="CatalogueEntryResponseMessage'' element=''p1 :CatalogueEntryResponseMessage'' />
</wsdl:message>
<wsdl:portType name="CatalogueEntryByWeightInterfaceOut">
<wsdl:operation name="CatalogueEntryByWeightInterfaceOut">
<wsdl:input message="p1:CatalogueEntryByWeightRequestMessage" />
<wsdl:output message="p1:CatalogueEntryResponseMessage'' />
</wsdl:operation>
</xsd:element>
</wsdl:portType>
</wsdi:definitions


<wsdl:element name="ChangeDetailsOfCitizenRequestMessage" type="p1:ChangeDetailsOfCitizenRequestType'' />
</xsd:element>
</xsd:complexType>
</xsd:schema>
</wsdl:element>
</xsd:complexType>
</xsd:schema>
</wsdl:types>
<wsdl:message name="CatalogueEntryByWeightInterfaceOut">
<wsdl:part name="CatalogueEntryByWeightInterfaceOut'' element=''p1 :CatalogueEntryByWeightRequestMessage'' />
</wsdl:message>
<wsdl:message name="CatalogueEntryResponseMessage">
<wsdl:part name="CatalogueEntryResponseMessage'' element=''p1 :CatalogueEntryResponseMessage'' />
</wsdl:message>
<wsdl:portType name="CatalogueEntryByWeightInterfaceOut">
<wsdl:operation name="CatalogueEntryByWeightInterfaceOut">
<wsdl:input message="p1:CatalogueEntryByWeightRequestMessage" />
<wsdl:output message="p1:CatalogueEntryResponseMessage'' />
</wsdl:operation>
</xsd:element>
</wsdl:portType>
</wsdl:binding>
</wsdl:service>
</wsdl:definitions>

<wsdl:service name="CatalogueEntryByWeightInterfaceOutService">
<xsd:appinfo source="http://sap.com/xi/TextID">
2ed766b0d7f811d9865ccb1d9b3831cc
</xsd:appinfo>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
</wsdl:types>
<wsdl:message name="ChangeDetailsOfCitizenRequestMessage">
<wsdl:part name="ChangeDetailsOfCitizenRequestMessage" element="pl:ChangeDetailsOfCitizenRequestMessage" />
</wsdl:message>
<wsdl:message name="ChangeDetailsOfCitizenResponseMessage">
<wsdl:part name="ChangeDetailsOfCitizenResponseMessage" element="pl:ChangeDetailsOfCitizenResponseMessage" />
</wsdl:message>
<wsdl:portType name="ChangeDetailsOfCitizenInterfaceOut">
<wsdl:operation name="ChangeDetailsOfCitizenInterfaceOut">
<wsdl:input message="pl:ChangeDetailsOfCitizenRequestMessage" />
<wsdl:output message="pl:ChangeDetailsOfCitizenResponseMessage" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ChangeDetailsOfCitizenInterfaceOutBinding" type="pl:ChangeDetailsOfCitizenInterfaceOut">
<wsdl:operation name="ChangeDetailsOfCitizenInterfaceOut">
<wsdl:input>
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdI/soap/" use="literal" />
</wsdl:input>
<wsdl:output>
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdI/soap/" use="literal" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="ChangeDetailsOfCitizenInterfaceOutService">
<wsdl:port name="ChangeDetailsOfCitizenInterfaceOutPort" binding="pl:ChangeDetailsOfCitizenInterfaceOutBinding">
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
</xsd:schema>
</xsd:element>
</xsd:baseType>
</xsd:complexType>
</xsd:schema>
</xsd:annotatio
Appendix H  Semantic Descriptions used in the Use case on Telecommunications

This appendix contains the BPMO instance of the Content Provision business process (Section H.1) as well as the resulting sBPEL instance from the BPMO2SBPEL translation (Section H.2) created for the Digital Content Downloading scenario, as presented in the use case on Telecommunications (Chapter 6).

H.1  BPMO instance for the Content Provision process

namespace { "http://ip-super.org/examples/process/bpmo/v2.0.1/examples#" ,
wsmostudio "http://www.wsmostudio.org#",
dc "http://purl.org/dc/elements/1.1#",
upo "http://ip-super.org/ontologies/process/upo/v2.0.1#",
bpmo "http://ip-super.org/ontologies/process/bpmo/v2.0.1#" }

ontology ContentProvision
nonFunctionalProperties
wsmostudio#version hasValue "0.7.3.2"
dc#description hasValue "Example of BPMO"
hasValue "SDatc: 13/10/2008" hasValue "Content Provision BPMO instance"
endNonFunctionalProperties

importsOntology
( "http://ip-super.org/ontologies/process/bpmo/v2.0.1#bpmo", "http://ip-super.org/ontologies/process/upo/v2.0.1#upo")

instance Process ContentProvision memberOf bpmo#Process
bpmo#hasName hasValue "Content Provision" bpmo#hasWorkflow hasValue Workflow_1207237301424_1670399772 bpmo#hasWSdescription hasValue SemanticCapability_Process_ContentProvision_WSMO upo#belongsToOrganisation hasValue telefonica upo#hasInvolvedRole hasValue serviceProvider

instance serviceProvider memberOf bpmo#BusinessRole
bpmo#hasName hasValue "Service Provider" bpmo#hasOrganisation hasValue telefonica

instance telefonica memberOf upo#Organisation upo#hasName hasValue "Telefonica"

instance sap memberOf upo#Organisation upo#hasName hasValue "SAP"

instance SemanticCapability_Process_ContentProvision_WSMO memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/wsno/ws/Process_ContentProvision#Process_ContentProvision"

instance Workflow_1207237301424_1670399772 memberOf bpmo#Workflow
Appendix H

instance Sequence_1207237301424_469283601 memberOf bpmo#Sequence
  bpmo#hasSize hasValue 5
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasOrderedElement hasValue {OrderedElement_1207826600209_370998577,
    OrderedElement_1207826600209_2134199540, OrderedElement_1207826600209_392991671,
    OrderedElement_1207826600209_773665020, OrderedElement_1207826600209_2001076281 }

instance OrderedElement_1207826600209_370998577 memberOf bpmo#OrderedElement
  bpmo#hasElement hasValue Receive_ContentRequest
  bpmo#hasOrder hasValue 1
  bpmo#hasNextElement hasValue OrderedElement_1207826600209_2001076281

instance Receive_ContentRequest memberOf bpmo#Receive
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasName hasValue "Receive Content Request"
  bpmo#hasPartnerWebService hasValue SemanticCapability_ContentRequester_WSMO
  bpmo#hasPartnerRole hasValue contentRequester
  bpmo#hasSendCounterpart hasValue SendContentResponse
  bpmo#hasInputDescription hasValue SemanticCapability_ContentRequestMessage
  bpmo#providesCapability hasValue SemanticCapability_ContentRequestOperation

instance contentRequester memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Content Requester"
  bpmo#hasOrganisation hasValue sap

instance SemanticCapability_ContentRequester_WSMO memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision/wsmo/RequestContentWS#RequestContentWS"

instance SemanticCapability_ContentRequestMessage memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision/wsmo/RequestContentWS#contentRequestMessage"

instance SemanticCapability_ContentRequestOperation memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision/wsmo/RequestContentWS#RequestContent"

instance OrderedElement_1207826600209_2001076281 memberOf bpmo#OrderedElement
  bpmo#hasElement hasValue MediationTask_PreparesRequest
  bpmo#hasOrder hasValue 2
  bpmo#hasNextElement hasValue OrderedElement_1207826600209_2134199540

instance OrderedElement_1207826600209_2134199540 memberOf bpmo#OrderedElement
  bpmo#hasElement hasValue ParallelSplitSynchronise_1207237301424_1641434797
  bpmo#hasOrder hasValue 3
  bpmo#hasNextElement hasValue OrderedElement_1207826600209_773665020

instance ParallelSplitSynchronise_1207237301424_1641434797 memberOf bpmo#ParallelSplitSynchronise
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasBranch hasValue {UnconditionalBranch_1207237301424_310422959,
    UnconditionalBranch_1207237301424_819276900 }

instance UnconditionalBranch_1207237301424_310422959 memberOf bpmo#UnconditionalBranch
  bpmo#hasElement hasValue GoalTask_GenerateURL

instance GoalTask_GenerateURL memberOf bpmo#GoalTask
  bpmo#hasName hasValue "Generate URL"
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasPartnerGoal hasValue SemanticCapability_GenerateURL_WSMO
  bpmo#hasPartnerRole hasValue contentProvider
  bpmo#hasInputDescription hasValue SemanticCapability_ContentURLReq
  bpmo#hasOutputDescription hasValue SemanticCapability_ContentURLRes

instance contentProvider memberOf bpmo#BusinessRole
  bpmo#hasName hasValue "Content Provider"
  bpmo#hasOrganisation hasValue telefonica

353
instance SemanticCapability_GenerateURL_WSMO memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision/wsmo/GoalGetURL#GoalGetURL"

instance SemanticCapability_ContentURLReq memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#reqGetURL"

instance SemanticCapability_ContentURLRes memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#resGetURL"

instance UnconditionalBranch_1207237301424 819276900 memberOf bpmo#UnconditionalBranch
bpmo#hasElement hasValue GoalTask_GenerateLicense

instance GoalTask_GenerateLicense memberOf bpmo#GoalTask
bpmo#hasName hasValue "Generate License"
bpmo#hasHomeProcess hasValue ProcessContentProvision
bpmo#hasPartnerGoal hasValue SemanticCapability_GenerateLicense_WSMO
bpmo#hasPartnerRole hasValue licenseProvider
bpmo#hasInputDescription hasValue SemanticCapabilityContentLicenseReq
bpmo#hasOutputDescription hasValue SemanticCapabilityContentLicenseRes

instance licenseProvider memberOf bpmo#BusinessRole
bpmo#hasName hasValue "License Provider"
bpmo#hasOrganisation hasValue telefonica

instance SemanticCapability_GenerateLicense_WSMO memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision/wsmo/GoalGetLicense#GoalGetLicense"

instance SemanticCapability_ContentLicenseReq memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#reqGetLicense"

instance SemanticCapability_ContentLicenseRes memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#resGetLicense"

instance OrderedElement_1207826600209_773665020 memberOf bpmo#OrderedElement
bpmo#hasElement hasValue MediationTaskAggregateResult
bpmo#hasOrder hasValue 4
bpmo#hasNextElement hasValue OrderedElement_1207826600209_392991671

instance OrderedElement_1207826600209_392991671 memberOf bpmo#OrderedElement
bpmo#hasElement hasValue SendContentResponse
bpmo#hasOrder hasValue 5

instance Send_ContentResponse memberOf bpmo#Send
bpmo#hasHomeProcess hasValue ProcessContentProvision
bpmo#hasName hasValue "Send Content Reply"
bpmo#hasPartnerWebService hasValue SemanticCapability_ContentRequester_WSMO
bpmo#hasPartnerRole hasValue contentRequester
bpmo#hasInputDescription hasValue SemanticCapability_ContentRequester_WSMO
bpmo#hasOutputDescription hasValue SemanticCapability_ContentResponseMessage
bpmo#requestsCapability hasValue SemanticCapability_ContentResponseOperation

instance SemanticCapability_ContentResponseMessage memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision/wsmo/SendContentWS#contentResponseMessage"

instance SemanticCapability_ContentResponseOperation memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision/wsmo/SendContentWS#SendContent"

instance MediationTask_PrepareRequest memberOf bpmo#MediationTask
bpmo#hasName hasValue "Prepare Request"
bpmo#hasHomeProcess hasValue ProcessContentProvision
bpmo#hasDataMediator hasValue {DataMediator_CopyRequestToURLReq, DataMediator_CopyRequestToLicenseReq}

instance DataMediator_CopyRequestToURLReq memberOf bpmo#DataMediator
bpmo#hasMediationService hasValue SemanticCapability_CopyURLReq_MediationService
bpmo#hasInputDescription hasValue SemanticCapability_ContentRequestMessage
bpmo#hasOutputDescription hasValue SemanticCapabilityContentLicenseRes

instance SemanticCapability_CopyURLReq_MediationService memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#mediationService1"
instance DataMediator_CopyRequestToLicenseReq memberOf bpmo#DataMediator
  bpmo#hasMediator hasValue SemanticCapability_CopyLicenseReq_Mediator
  bpmo#hasInputDescription hasValue SemanticCapability_ContentRequestMessage
  bpmo#hasOutputDescription hasValue SemanticCapability_ContentLicenseReq

instance SemanticCapability_CopyLicenseReq_Mediator memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#mediator2"

instance MediationTask_AggregateResult memberOf bpmo#MediationTask
  bpmo#hasName hasValue "Aggregate Result"
  bpmo#hasHomeProcess hasValue Process_ContentProvision
  bpmo#hasDataMediator hasValue {DataMediator_CopyURLToResponse, DataMediator_CopyLicenseToResponse}

instance DataMediator_CopyURLToResponse memberOf bpmo#DataMediator
  bpmo#hasMediator hasValue SemanticCapability_CopyURL_Mediator
  bpmo#hasInputDescription hasValue SemanticCapability_ContentURLReq
  bpmo#hasOutputDescription hasValue SemanticCapability_ContentResponseMessage

instance DataMediator_CopyLicenseToResponse memberOf bpmo#DataMediator
  bpmo#hasMediator hasValue SemanticCapability_CopyLicense_Mediator
  bpmo#hasInputDescription hasValue SemanticCapability_ContentLicenseReq
  bpmo#hasOutputDescription hasValue SemanticCapability_ContentLicenseResponseMessage

instance SemanticCapability_CopyLicense_Mediator memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#mediator3"

instance DataMediator_CopyLicenseToResponse memberOf bpmo#DataMediator
  bpmo#hasMediator hasValue SemanticCapability_CopyLicense_Mediator
  bpmo#hasInputDescription hasValue SemanticCapability_ContentLicenseReq
  bpmo#hasOutputDescription hasValue SemanticCapability_ContentLicenseResponseMessage

instance SemanticCapability_CopyLicense_Mediator memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://ip-super.org/sws/ContentProvision#mediator4"

H.2 sBPEL instance for the Content Provision Process

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace _"http://ip-super.org/instances/auto-generated/sBPEL#" 
  sbpel _"http://ip-super.org/ontologies/process/sbpel/v2.0.0#",
    bpmo _"http://ip-super.org/ontologies/process/bpel20/v2.0.0#",
    wsmostudio _"http://www.wsmostudio.org#"

ontology ContentProvision_sBPEL
  nonFunctionalProperties
    wsmo#version hasValue "0.7.3.2"
    _"http://purl.org/dc/elements/1.1#creator" hasValue "Automatically generated by BPM02sBPEL Tool"
endNonFunctionalProperties

importsOntology
  { _"http://ip-super.org/ontologies/process/sbpel/v2.0.0#sbpel",
    _"http://ip-super.org/ontologies/process/bpel20/v2.0.0#bpel20",
    _"http://ip-super.org/examples/process/bpmo/v2.0.1/examples#ContentProvision"
  }

instance Process_ContentProvision_sBPEL memberOf sbpel#SemanticProcess
  sbpel#hasName hasValue "contentProvision"
  sbpel#hasTargetNamespace hasValue _"http://ip-super.org/instances/auto-generated/sBPEL#",
  sbpel#hasPartner hasValue contentRequester_sBPEL
  sbpel#hasConversation hasValue [GoalTask_GenerateURL_sBPELConversation,
    GoalTask_GenerateLicense_sBPELConversation]
  sbpel#hasVariable hasValue {SemanticCapability_ContentRequestMessage_sBPEL,
    SemanticCapability_ContentURLRes_sBPEL, SemanticCapability_ContentLicenseReq_sBPEL,
    SemanticCapability_ContentLicenseRes_sBPEL, SemanticCapability_ContentRequestMessage_sBPEL,
    SemanticCapability_ContentURLReq_sBPEL, GoalTask_GenerateURL_sBPEL, GoalTask_GenerateLicense_sBPEL}

instance Process_ContentProvision_sBPEL memberOf sbpel#SemanticProcess
  sbpel#hasName hasValue "contentProvision"
  sbpel#hasTargetNamespace hasValue _"http://ip-super.org/instances/auto-generated/sBPEL#",
  sbpel#hasPartner hasValue contentRequester_sBPEL
  sbpel#hasConversation hasValue [GoalTask_GenerateURL_sBPELConversation,
    GoalTask_GenerateLicense_sBPELConversation]
  sbpel#hasVariable hasValue {SemanticCapability_ContentRequestMessage_sBPEL,
    SemanticCapability_ContentURLRes_sBPEL, SemanticCapability_ContentLicenseReq_sBPEL,
    SemanticCapability_ContentLicenseRes_sBPEL, SemanticCapability_ContentRequestMessage_sBPEL,
    SemanticCapability_ContentURLReq_sBPEL, GoalTask_GenerateURL_sBPEL, GoalTask_GenerateLicense_sBPEL}
```

355
instance Sequence_1207237301424_469283601_sBPEL memberOf bpel#Sequence
bpel#hasOrderedActivity hasValue OrderedElement_1207826600209_370998577_sBPEL
bpel#correspondsTo hasValue _"http://ip-super.org/examples/process/bpmo/v2.0.1/examples#Sequence_1207237301424_469283601"

instance OrderedElement_1207826600209_370998577_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue Receive_ContentRequest_sBPEL
bpel#hasActivity hasValue Receive_ContentRequest_sBPELReceive

instance contentRequester_sBPEL memberOf sbpel#Partner
sbpel#hasName hasValue "Content Requester"
sbpel#hasBusinessEntity hasValue "SAP"
sbpel#hasConversation hasValue Receive_ContentRequest_sBPELConversation

instance SemanticCapability_ContentRequestMessage_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_ContentRequestMessage_sBPEL"
bpel#hasType hasValue SemanticCapability_ContentRequestMessage_sBPEL_WSDLMessage
sbpel#hasSemanticType hasValue _"http://ip-super.org/sws/ContentProvision/WSMO/RequestContentWS#ContentRequestMessage"

instance OrderedElement_1207826600209_2001076281_sBPEL memberOf bpel#OrderedActivity
instance Receive_ContentRequest_sBPEL memberOf sbpel#ExtensionActivity

instance ParallelSplitSynchronise_1207237301424_1641434797_sBPEL memberOf bpel#Flow
bpel#correspondsTo hasValue _"http://ip-super.org/examples/process/bpmo/v2.0.1/examples#ParallelSplitSynchronise_1207237301424_1641434797"

instance GoalTask_GenerateURL_sBPEL memberOf bpel#ExtensionActivity
bpel#correspondsTo hasValue _"http://ip-super.org/examples/process/bpmo/v2.0.1/examples#GoalTask_GenerateURL"

instance GoalTask_GenerateLicense_sBPEL memberOf bpel#ExtensionActivity
bpel#correspondsTo hasValue _"http://ip-super.org/examples/process/bpmo/v2.0.1/examples#GoalTask_GenerateLicense"
Appendix H

instance SendContentResponse_sBPEL memberOf bpel#ExtensionActivity
bpel#hasActivity hasValue "Send Content Response"
bpel#correspondsTo hasValue "http://ip-superv.org/examples/process/bpmo/v2.0.1/examples#SendContentResponse"
bpel#hasActivity hasValue SendContentResponse_sBPELSend

instance SemanticCapability_ContentResponseMessage_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_ContentResponseMessage_sBPEL"
bpel#hasType hasValue SemanticCapability_ContentResponseMessage_sBPEL
sbpel#hasSemanticType hasValue "http://ip-superv.org/sws/Content Provision/wsmo/ContentResponseMessage"

instance MediationTask_PrepareRequest_sBPEL memberOf bpel#Assign
bpel#hasName hasValue "Prepare Request"
bpel#hasAssignOperation hasValue (DataMediator_CopyRequestToURLRequest_sBPEL,
DataMediator_CopyRequestToLicenseReq_sBPEL)
bpel#hasAssignOperation hasValue "http://ip-superv.org/examples/process/bpmo/v2.0.1/examples#MediationTask_PrepareRequest"

instance DataMediator_CopyRequestToURLRequest_sBPEL memberOf sbpel#Mediate
sbpel#usesDataMediator hasValue SemanticCapability_CopyURLRequest_sBPEL
sbpel#hasInputVariable hasValue SemanticCapability_RequestMessage_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_RequestMessage_sBPEL

instance DataMediator_CopyRequestToLicenseReq_sBPEL memberOf sbpel#Mediate
sbpel#usesDataMediator hasValue SemanticCapability_CopyLicenseReq_sBPEL
sbpel#hasInputVariable hasValue SemanticCapability_RequestMessage_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_RequestMessage_sBPEL

instance MediationTask_AggregateResult_sBPEL memberOf bpel#Assign
bpel#hasName hasValue "Aggregate Result"
bpel#hasAssignOperation hasValue (DataMediator_CopyURLToResponse_sBPEL,
DataMediator_CopyLicenseToResponse_sBPEL)
bpel#hasAssignOperation hasValue "http://ip-superv.org/examples/process/bpmo/v2.0.1/examples#MediationTask_AggregateResult"

instance DataMediator_CopyURLToResponse_sBPEL memberOf sbpel#Mediate
sbpel#usesDataMediator hasValue SemanticCapability_CopyURL_sBPEL
sbpel#hasInputVariable hasValue SemanticCapability_ContentURLReq_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_ContentResponseMessage_sBPEL

instance DataMediator_CopyLicenseToResponse_sBPEL memberOf sbpel#Mediate
sbpel#usesDataMediator hasValue SemanticCapability_CopyLicense_sBPEL
sbpel#hasInputVariable hasValue SemanticCapability_ContentLicenseReq_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_ContentResponseMessage_sBPEL

instance ReceiveContentRequest_sBPELReceive memberOf sbpel#Receive
sbpel#hasName hasValue "Receive Content Request"
sbpel#hasVariable hasValue SemanticCapability_RequestMessage_sBPEL
sbpel#belongsToConversation hasValue ReceiveContentRequest_sBPELConversation

instance GoalTask_GenerateURL_sBPELSendReceive memberOf sbpel#SendReceive
sbpel#hasName hasValue "Generate URL"
sbpel#hasInputVariable hasValue SemanticCapability_ContentURLReq_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_ContentURLRes_sBPEL
sbpel#belongsToConversation hasValue GoalTask_GenerateURL_sBPELConversation

instance GoalTask_GenerateLicense_sBPELSendReceive memberOf sbpel#SendReceive
sbpel#hasName hasValue "Generate License"
sbpel#hasInputVariable hasValue SemanticCapability_ContentLicenseReq_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_ContentLicenseRes_sBPEL
sbpel#belongsToConversation hasValue GoalTask_GenerateLicense_sBPELConversation

instance SendContentResponse_sBPEL Send memberOf sbpel#Send
sbpel#hasName hasValue "Send Content Reply"
sbpel#hasVariable hasValue SemanticCapability_RequestMessage_sBPEL
sbpel#belongsToConversation hasValue SendContentResponse_sBPELConversation

instance ReceiveContentRequest_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "ReceiveContentRequest_sBPELConversation"
instance GoalTask_GenerateURLs_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "GoalTask_GenerateURLs_sBPELConversation"
sbpel#describesInterface hasValue GoalTask_GenerateURLs_sBPELOutgoingInterface

instance GoalTask_GenerateLicense_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "GoalTask_GenerateLicense_sBPELConversation"
sbpel#describesInterface hasValue GoalTask_GenerateLicense_sBPELOutgoingInterface

instance Receive_ContentRequest_sBPELInterface memberOf sbpel#IncomingInterface
sbpel#hasWebServiceDescription hasValue "http://ip-super.org/sws/Content Provision/wsmo/RequestContentWS#RequestContentWS"

instance GoalTask_GenerateURLs_sBPELOutgoingInterface memberOf sbpel#OutgoingInterface
sbpel#hasGoalDescription hasValue "http://ip-super.org/sws/Content Provision/wsmo/GoalGetURL#GoalGetURL"

instance GoalTask_GenerateLicense_sBPELOutgoingInterface memberOf sbpel#OutgoingInterface
sbpel#hasGoalDescription hasValue "http://ip-super.org/sws/Content Provision/wsmo/GoalGetLicense#GoalGetLicense"

instance SemanticCapability_CopyURLReq_MediationService_sBPELDataMediator memberOf sbpel#DataMediator
sbpel#hasMediationService hasValue "http://ip-super.org/sws/Content Provision#wsmo/mediationService1"

instance SemanticCapability_CopyLicenseReq_MediationService_sBPELDataMediator memberOf sbpel#DataMediator
sbpel#usesMediationService hasValue "http://ip-super.org/sws/Content Provision#wsmo/mediationService2"

instance SemanticCapability_CopyURL_MediationService_sBPELDataMediator memberOf sbpel#DataMediator
sbpel#usesMediationService hasValue "http://ip-super.org/sws/Content Provision#wsmo/mediationService3"

instance SemanticCapability_CopyLicense_MediationService_sBPELDataMediator memberOf sbpel#DataMediator
sbpel#usesMediationService hasValue "http://ip-super.org/sws/Content Provision#wsmo/mediationService4"

instance SemanticCapability_ContentRequestMessage_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_ContentURLReq_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_ContentURLRes_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_ContentLicenseReq_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_ContentLicenseRes_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_ContentResponseMessage_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""
Appendix I  Semantic Descriptions used in the SWS Challenge Use case

This appendix contains the mediation scenario presented in the SWS Challenge (sws-challenge.org/wiki), the BPMO instance created for the mediator process, sample data mappings used, the resulting sBPEL and the WSDL created for the mediator process.

I.1  Purchase Order Mediation Scenario

This picture has been copied here from the SWS Challenge website for convenience.
1.2 BPMO instance for the Mediator Process

This BPMO instance was created from the diagram generated with WSMO Studio for the SWS Challenge Mediator Process as presented in Chapter 5, Section 5.4.2.

namespace { "http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#" 
  wsmostudio _"http://www.wsmostudio.org#", 
  dc _"http://purl.org/dc/elements/1.1#", 
  upo _"http://ip-super.org/ontologies/process/upo/v2.0.1#", 
  bpmo _"http://ip-super.org/ontologies/process/bpmo/v2.0.1#" } 

ontology MoonMediatorProcessInstance
  nonFunctionalProperties
    wsmostudio#version hasValue "0.7.3.2"
    hasValue "Liliana Cabral"
    hasValue "KMI. The Open University" 
    wsmostudio#version hasValue "0.7.3.2"
    hasValue "Use Case"
    dc#description hasValue "BPMO instances for the Moon Mediator Process in the SWS Challenge"
    dc#language hasValue "en-GB"
    dc#date hasValue "SDate: 13/10/2008$"
    bpmo hasValue "MoonMediatorProcess BPMO instance"
  endNonFunctionalProperties

importsOntology
  { _"http://ip-super.org/ontologies/process/bpmo/v2.0.1#bpmo", 
    _"http://ip-super.org/ontologies/process/upo/v2.0.1#upo")

instance Process_MoonMediator memberOf bpmo#Process 
bpmo#hasName hasValue "Moon Mediator Process"
bpmo#hasWorkflow hasValue Workflow_1217869140038_181603276 
upo#hasInvolvedRole hasValue moonMediator

instance moonMediator memberOf bpmo#BusinessRole 
bpmo#hasName hasValue "Moon Mediator"
bpmo#hasOrganisation hasValue kmi

instance kmi memberOf upo#Organisation 
upo#hasName hasValue "KMi, The Open University, UK"

instance blue memberOf upo#Organisation 
upo#hasName hasValue "Blue, SWS Challenge"

instance customer memberOf bpmo#BusinessRole 
bpmo#hasName hasValue "Blue Customer"
bpmo#hasOrganisation hasValue blue

instance moon memberOf upo#Organisation 
upo#hasName hasValue "Moon, SWS Challenge"

instance moonCRM memberOf bpmo#BusinessRole 
bpmo#hasName hasValue "Moon Customer Relationship Management"
bpmo#hasOrganisation hasValue moon

instance moonOM memberOf bpmo#BusinessRole 
bpmo#hasName hasValue "Moon Order Management"
instance Workflow_1217869140038_181603276 memberOf bpmo#Workflow
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasFirstWorkflowElement hasValue StartEvent_1

instance StartEvent_1 memberOf bpmo#StartEvent
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasName hasValue "Start"

instance ControlflowConnector_100 memberOf bpmo#ControlflowConnector
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasSource hasValue StartEvent_1
bpmo#hasTarget hasValue Receive_PO

instance Receive_PO memberOf bpmo#Receive
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasName hasValue "Receive Purchase Order"
bpmo#hasPartnerWebService hasValue SemanticCapability_Receive_PO_WSMO
bpmo#hasPartnerRole hasValue customer
bpmo#hasSendCounterpart hasValue Send_Send_PO_Conf
bpmo#hasInputDescription hasValue SemanticCapability_PurchaseOrderDesc

instance SemanticCapability_PurchaseOrderDesc memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator/PurchaseOrderRequest"

instance SemanticCapability_Receiv PO_WSMO memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/RequestPOWS#RequestPOWS"

instance ControlflowConnector_200 memberOf bpmo#ControlflowConnector
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasSource hasValue Receive_PO
bpmo#hasTarget hasValue MediationTask_MapPurchaseOrder

instance MediationTask_MapPurchaseOrder memberOf bpmo#MediationTask
bpmo#hasName hasValue "Map Purchase Order"
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasDataMediator hasValue [DataMediator_MapOrderRequestToSearchCustomer,
DataMediator_MapOrderRequestToOrder]

instance SemanticCapability_MapOrderRequestToSearchCustomer memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator#SearchCustomerFromPurchaseOrderRequest"

instance SemanticCapability_MapOrderRequestToOrder memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator#OrderFromPurchaseOrderRequest"

instance ControlflowConnector_300 memberOf bpmo#ControlflowConnector
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasSource hasValue MediationTask_MapPurchaseOrder
bpmo#hasTarget hasValue GoalTask_SearchCustomer

instance GoalTask_SearchCustomer memberOf bpmo#GoalTask
bpmo#hasName hasValue "Search Customer"
bpmo#hasPartnerGoal hasValue SemanticCapability_SearchCustomer_WSMO
bpmo#hasPartnerRole hasValue moonOM
bpmo#hasInputDescription hasValue SemanticCapability_SearchCustomerDesc
bpmo#hasOutputDescription hasValue SemanticCapability_SearchCustomerResponseDesc
instance SemanticCapability_SearchCustomerDesc memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/ usecase/swsc/datamediator#SearchCustomer"

instance SemanticCapability_SearchCustomerResponseDesc memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/ usecase/swsc/datamediator#SearchCustomerResponse"

instance SemanticCapability_SearchCustomer_WSMO memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue
"http://lilianathesis.org/ usecase/swsc/wsmo/SearchCustomerGoal#SearchCustomerGoal"

instance ControlflowConnector_400 memberOf bpmo#ControlflowConnector
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasSource hasValue GoalTask_SearchCustomer
bpmo#hasTarget hasValue GoalTask_CreateOrder

instance GoalTaskCreateOrder inemberOf bpmo#GoalTask
bpmo#hasName hasValue "Create Order"
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasPartnerGoal hasValue SemanticCapability_CreateOrder_WSMO
bpmo#hasPartnerRole hasValue moonCRM
bpmo#hasInputDescription hasValue SemanticCapability_OrderDesc
bpmo#hasOutputDescription hasValue SemanticCapability_OrderResponseDesc

instance ControlflowConnector_500 memberOf bpmo#ControlflowConnector
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasSource hasValue GoalTask_CreateOrder
bpmo#hasTarget hasValue Repeat_AddLineItem

instance Repeat_AddLineItem memberOf bpmo#Repeat
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasCondition hasValue Condition_1218020116299_1731096821
bpmo#executes hasValue Send_AddLineItem

instance Condition_1218020116299_1731096821 memberOf bpmo#Condition
bpmo#hasExpression hasValue "Another Item?"

instance Send_AddLineItem memberOf bpmo#Send
bpmo#hasName hasValue "Add Line Item"
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasPartnerWebService hasValue SemanticCapability_AddLineItem_WSMO
bpmo#hasPartnerRole hasValue moonCRM
bpmo#hasReceiveCounterpart hasValue Receive_ConfirmLineItem
bpmo#hasOutputDescription hasValue SemanticCapability_LineItemDesc

instance SemanticCapability_LineItemDesc memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/ usecase/swsc/datamediator#LineItem"

instance SemanticCapability_AddLineItem_WSMO memberOf bpmo#SemanticCapability
bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/ usecase/swsc/wsmo/AddLineItemWS#AddLineItemWS"

instance ControlflowConnector_600 memberOf bpmo#ControlflowConnector
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasSource hasValue GoalTask_CreateOrder
bpmo#hasTarget hasValue GoalTask_CloseOrder

instance GoalTask_CloseOrder memberOf bpmo#GoalTask
bpmo#hasName hasValue "Close Order"
bpmo#hasHomeProcess hasValue Process_MoonMediator
bpmo#hasPartnerGoal hasValue SemanticCapability_CloseOrder_WSMO
bpmo#hasPartnerRole hasValue moonCRM


bpmo\#hasInputDescription hasValue SemanticCapability\_CloseOrderDesc
bpmo\#hasOutputDescription hasValue SemanticCapability\_CloseOrderResponseDesc

instance SemanticCapability\_CloseOrderDesc memberOf bpmo\#SemanticCapability
bpmo\#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator\#CloseOrder"

instance SemanticCapability\_CloseOrderResponseDesc memberOf bpmo\#SemanticCapability
bpmo\#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator\#CloseOrderResponse"

instance SemanticCapability\_CloseOrder\_WSMO memberOf bpmo\#SemanticCapability
bpmo\#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/CloseOrderGoal\#CloseOrderGoal"

instance ControlflowConnector\_700 memberOf bpmo\#ControlflowConnector
bpmo\#hasHomeProcess hasValue Process\_MoonMediator
bpmo\#hasSource hasValue GoalTask\_CloseOrder
bpmo\#hasTarget hasValue Repeat\_ConfirmLineItem

instance Condition\_1218020116299\_218215930 memberOf bpmo\#Condition
bpmo\#hasExpression hasValue "Another Item?"

instance Receive\_ConfirmLineItem memberOf bpmo\#Receive
bpmo\#hasName hasValue "Confirm Line Item"

instance Receive\_ConfirmLineItemDesc memberOf bpmo\#SemanticCapability
bpmo\#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator\#OrderLineItemConfirmation"

instance DataMediator\_MapRequestToConfirmation memberOf bpmo\#DataMediator
bpmo\#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator\#ConfirmationFromPurchaseOrderRequest"

instance Send\_SendPOConf memberOf bpmo\#Send
bpmo\#hasName hasValue "Send PO Confirmation"

APPENDIX I

instance SemanticCapability_POConfirmationDesc memberOf bpmo#SemanticCapability
  bpmo#hasSemanticDescription hasValue "http://lilianathesis.org/usecase/swsc/datamediator#PurchaseOrderConfirmation"

instance ControlFlowConnector_1000 memberOf bpmo#ControlFlowConnector
  bpmo#hasHomeProcess hasValue Process_MoonMediator
  bpmo#hasSource hasValue Send_SendPOConf
  bpmo#hasTarget hasValue EndEvent_1

instance EndEvent_1 memberOf bpmo#EndEvent
  bpmo#hasHomeProcess hasValue Process_MoonMediator
  bpmo#hasName hasValue "End"

1.3 Sample Data Mappings for the Mediator Process

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
  dc _"http://purl.org/dc/elements/1.1/" }

ontology dataMappings
  non FunctionalProperties
    wsmostudio#version hasValue "0.7.3.2"
    hasValue "Liliana Cabral"
    dc:pubUihi: hasValue "KMI, The Open University"
    wsmostudio#version hasValue "0.7.3.2"
    hasValue "Example of data mappings"
    dc:description hasValue "Simplified concepts for SWSChallenge to be used with http://lilianathesis.org/usecase/swsc/bpmo/MoonMediat0rProcessInstance#"
    dc:language hasValue "en-GB"
    hasValue "SDate: 13/10/2008"
    atr:rule hasValue "MoonMediatorProcess dataMappings"
endNonFunctionalProperties

/*
* Receive Purchase Order
*/

http://sws-challenge.org/schemas/met/3A4_Simplified_PurchaseOrderRequest.xsd”>
  <core:fromRole>
    <core:PartnerRoleDescription>
      <core:ContactInformation>
        <core:contactName>
          <core:FreeFormText>Stefan Blue</core:FreeFormText>
          <core:contactName>
            <core:EmailAddress>stefan.blue@blue.com</core:EmailAddress>
            <core:telephoneNumber>
              <core:CommunicationsNumber>+43(650)89930011</core:CommunicationsNumber>
              <core:telephoneNumber>
                <core:ContactInformation>
                  <dict:GlobalPartnerRoleClassificationCode>Buyer</dict:GlobalPartnerRoleClassificationCode>
                  <core:PartnerDescription>
                    <core:BusinessDescription>
                      <core:businessName>
                        <core:FreeFormText>Blue Company</core:FreeFormText>
                        <core:businessName>
                          <core:BusinessDescription>
                            */

concept PurchaseOrderRequest
  fromRole oType PartnerRoleDescription

364
hasPurchaseOrder ofType PurchaseOrder

concept PartnerRoleDescription
  hasContact ofType ContactInformation
  hasRole ofType _string
  partnerDescription ofType PartnerDescription

concept PurchaseOrder
  productLineItem ofType ProductLineItemDescription
  shipTo ofType PartnerDescription
  totalAmount ofType FinancialAmount

concept ProductLineItemDescription
  lineNumber ofType _integer
  orderQuantity ofType _integer
  productIdentification ofType _string
  requestedUnitPrice ofType FinancialAmount
  totalLineItemAmount ofType FinancialAmount

concept ContactInformation
  contactName ofType _string
  emailAddress ofType _string
  telephoneNumber ofType _string

concept PartnerDescription
  contactInfo ofType ContactInformation
  businessInfo ofType BusinessDescription
  physicalLocation ofType PhysicalAddress

concept BusinessDescription
  hasName ofType _string

concept PhysicalAddress
  addressLine1 ofType _string
  cityName ofType _string
  countryCode ofType _string
  postalCode ofType _string

concept FinancialAmount
  currencyCode ofType _string
  monetaryAmount ofType _string

/*
 * Search Customer
 */
<SearchCustomer xmlns="mooncompany">
  <searchString>Blue Company</searchString>
</SearchCustomer>

concept SearchCustomer
  searchString ofType _string

<ns1:SearchCustomerResponse xmlns="mooncompany" xmlns:ns1="mooncompany">
  <ns1:customerId>1</ns1:customerId>
</ns1:SearchCustomerResponse>

concept SearchCustomerResponse
  customerId ofType _string

/*
 * Map Purchase Order to SearchCustomer
 * Queries: MapOrderRequestToSearchCustomer(?request, ?sc)
 *   ?sc[searchString hasValue ?name] memberOf SearchCustomer
 */

relation MapOrderRequestToSearchCustomer ofType PurchaseOrderRequest
  impliesType SearchCustomer

axiom SearchCustomerFromPurchaseOrderRequest
  definedBy

365
?or[fromRole hasValue ?pr] memberOf PurchaseOrderRequest
and ?pr[partnerDescription hasValue ?pd] memberOf PartnerRoleDescription
and ?pd[businessInfo hasValue ?bd] memberOf PartnerDescription
and ?bd[name hasValue ?name] memberOf BusinessDescription
implies searchCustomer(?or)[searchString hasValue ?name] memberOf SearchCustomer
and MapOrderRequestToSearchCustomer(?or, searchCustomer(?or)).

/*
* Create Order
*/

/*
<Order xmlns="mooncompany">
<authToken>authToken</authToken>
<contact>
  <name>Stefan Blue</name>
  <telephone>+43(650)8993001</telephone>
  <email>stefan.blue@blue.com</email>
</contact>
<shipTo>
  <name>Blue Company</name>
  <street>North Business Center, Block 9</street>
  <city>Innsbruck</city>
  <postalCode>A-6020</postalCode>
  <country>AT</country>
</shipTo>
*/

concept Order
  authToken ofType string
  contact ofType Contact
  shipTo ofType OrderInformation
  billTo ofType OrderInformation

concept OrderInformation
  name ofType string
  street ofType string
  city ofType string
  postalCode ofType string
  country ofType string

concept Contact
  name ofType string
  telephone ofType string
  email ofType string

/*
* Map PurchaseOrder(Blue request) to Order(Moon)
* Queries: MapOrderRequestToOrder(?blueReq, ?moonOrder)
* ?order [authToken hasValue ?auth, contact hasValue ?c, shipTo hasValue ?s ] memberOf Order and
* ?s [name hasValue ?businessName, street hasValue ?street, city hasValue ?city, postalCode hasValue ?postalCode, country hasValue ?country] memberOf OrderInformation
* */

relation MapOrderRequestToOrder( impliesType PurchaseOrderRequest, impliesType Order)
axiom OrderFromPurchaseOrderRequest
definedBy
?request[fromRole hasValue ?pr] memberOf PurchaseOrderRequest
and ?pr[partnerDescription hasValue ?pd] memberOf PartnerRoleDescription
and ?pd[contactInfo hasValue ?ci, businessInfo hasValue ?bd, physicalLocation hasValue ?pl] memberOf PartnerDescription
and ?ci[contactName hasValue ?contactName] memberOf ContactInformation
and ?bd[hasName hasValue ?bussName] memberOf BusinessDescription
and ?pl[addressLine1 hasValue ?adr, cityName hasValue ?c, countryCode hasValue ?co, postalCode hasValue ?pc] memberOf PhysicalAddress
implies moonOrder(?request)[authToken hasValue "LilianaCabral", contact hasValue contact(?ci), shipTo hasValue shipTo(?bd)] memberOf Order
and contact(?ci)[name hasValue ?contactName] memberOf Contact
and shipTo(?bd)[name hasValue ?bussName, street hasValue ?adr, city hasValue ?c, postalCode hasValue ?postalCode, country hasValue ?country] memberOf OrderInformation
*/
postalCode hasValue ?pc, country hasValue ?co | memberOf OrderInformation
and MapOrderRequestToOrder(?request, moonOrder(?request)).

/*
<ns1:NewOrderResponse xmlns="mooncompany" xmlns:ns1="mooncompany">
  <ns1:orderId>9</ns1:orderId>
  <ns1:NewOrderResponse/>
*/

concept NewOrderResponse
  orderId ofType string

/*
* Add Line Item
*/
/*
<LineItem xmlns="mooncompany">
  <authToken>LilianaCabral</authToken>
  <orderId>496</orderId>
  <item>
    <articleId>myitem1</articleId>
    <quantity>1</quantity>
  </item>
</LineItem>
*/

concept LineItem
  authToken ofType string
  orderId ofType string
  item ofType ItemDescription

concept ItemDescription
  articleId ofType string
  quantity ofType integer

/*
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapen v: Body>
    <ns1:AddLineItemResponse xmlns="mooncompany" xmlns:ns1="mooncompany">
      <ns1:orderId>2</ns1:orderId>
      <ns1:itemId>4</ns1:itemId>
    </ns1:AddLineItemResponse>
  </soapen:Body>
</soapenv:Envelope>
*/

concept AddLineItemResponse
  orderId ofType string
  itemId ofType string

/*
* Close Order
*/
/*
<CloseOrder xmlns="mooncompany">
  <authToken>LilianaCabral</authToken>
  <orderId>496</orderId>
</CloseOrder>
*/

concept CloseOrder
  authToken ofType string
  orderId ofType string

/*
<ns1:CloseOrderResponse xmlns="mooncompany" xmlns:ns1="mooncompany">
  <ns1:orderId>2</ns1:orderId>
*/

367
concept CloseOrderResponse
orderld ofType string
itemsNo ofType integer
*/
* Confirm Line Item
*/
*/
<OrderLineItemConfirmation xmlns="mooncompany">
<authToken>LilianaCabral</authToken>
<orderId>496</orderId>
<itemld>4</itemld>
<status>pending</status>
</OrderLineItemConfirmation>
*/
concept OrderLineItemConfirmation
authToken ofType string
orderld ofType string
itemld ofType string
status ofType string
concept PurchaseOrderConfirmation
*/
* Map Result
*/
* Map PurchaseOrderRequest(Blue request) to PurchaseOrderConfirmation
* -- A PurchaseOrderConfirmation must have the same content as the PurchaseOrderRequest
* Queries: MapPurchaseOrderToConfirmation(BlueReq, ?confirm)
* ?confirmation memberOf PurchaseOrderConfirmation
* */
r
relation MapPurchaseOrderToConfirmation(impliesType PurchaseOrderRequest, impliesType PurchaseOrderConfirmation)

axiom ConfirmationFromPurchaseOrderRequest
definedBy
"purchaseOrderRequest memberOf PurchaseOrderRequest
implies confirmation(confirmation(purchaseOrderRequest))
and MapPurchaseOrderToConfirmation(purchaseOrderRequest, confirmation(purchaseOrderRequest))."*/
* Send PO Confirmation
*/
*/
http://www.sws-challenge.org/schemas/met/3A4_Simplified_PurchaseOrderConfirmation.xsd">
<core:fromRole>
<core:PartnerRoleDescription>
<core:ContactInformation>
<core:contactName>
<core:FreeFormText>Stefan Blue</core:FreeFormText>
</core:contactName>
<core:EmailAddress>stefan.blue@blue.com</core:EmailAddress>
<core:telephoneNumber>
*/

instance bluePORequest memberOf PurchaseOrderRequest
fromRole hasValue bluePartnerRole
hasPurchaseOrder hasValue bluePurchaseOrder
instance bluePartnerRole memberOf PartnerRoleDescription
hasContact hasValue blueContact
hasRole hasValue "Buyer"
partnerDescription hasValue bluePartnerDescription
instance bluePartnerDescription memberOf PartnerDescription
contactInfo hasValue blueContact
businessInfo hasValue blueBusiness
physicalLocation hasValue blueAddress

368
I.4 WSDL for the Mediator Process

Below is the WSDL file we created for the Mediator Process in section A.1. This WSDL file was generated automatically by our local Web Service implementation using the Spring Web Services framework (http://springframework.org/spring-ws). This Web Service implementation was done to validate and test the messages to be sent and received. The WSDL specifies the messages and operations used by the Web Service.
</xml version="1.0" encoding="UTF-8"/>

- <wsdl :definitions xmlns:wsdl= 'http://schemas.xmlsoap.org/wsdl/
 xmlns:schema="http://kmi.open.ac.uk/swsc/schemas/mediatorMessages
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
 xmlns:tns= "http://kmi.open.ac.uk/swsc/mediator/definitions"
 targetNamespace= "http://kmi.open.ac.uk/swsc/mediator/definitions">
- <wsdl :types>
 xmlns:xsd= "http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
 targetNamespace= "http://kmi.open.ac.uk/swsc/mediator/definitions">
 challenge.org/schemas/met/3A4_PurchaseOrder_Dictionary.xsd"/>
 challenge.org/schemas/met/3A4_PurchaseOrder_CoreElements.xsd"/>
- <xsd:element name="Pip3A4PurchaseOrderRequest">
  - <xsd:complexType>
    - <xsd:sequence>
      - <xsd:element ref="core:fromRole" />
      - <xsd:element ref="tns:PurchaseOrder" />
      - <xsd:element ref="core:thisDocumentGenerationDateTime" />
      - <xsd:element ref="core:thisDocumentIdentifier" />
      - <xsd:element ref="core:toRole" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
- <xsd:element name="PurchaseOrder">
  - <xsd:annotation>
    <xsd:documentation>The collection of business properties that describe a buyer's offer to purchase a quantity of products at an
 agreed price and schedule. Additionally to it the status of each item is provided.</xsd:documentation>
  </xsd:annotation>
  - <xsd:complexType>
    - <xsd:sequence>
      - <xsd:element minOccurs="0" ref="dict:GlobalConfirmationTypeCode" />
      - <xsd:element ref="dict:GlobalPurchaseOrderStatusCode" />
      - <xsd:element maxOccurs="unbounded" ref="dict:GlobalPurchaseOrderTypeCode" />
      - <xsd:element ref="core:isDropShip" />
      - <xsd:element ref="core:LineNumber" />
      - <xsd:element ref="tns:OrderQuantity" />
      - <xsd:element ref="core:ProductIdentification" />
      - <xsd:element ref="core:requestedEvent" />
      - <xsd:element minOccurs="0" ref="core:requestUnitPrice" />
      - <xsd:element maxOccurs="unbounded" minOccurs="0" ref="core:ShippedQuantityInformation" />
      - <xsd:element minOccurs="0" ref="core:totalLineItemAmount" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
- <xsd:element name="ProductLineItem">
  - <xsd:annotation>
    <xsd:documentation>The collection of business properties that describe a business document entry for a
 product.</xsd:documentation>
  </xsd:annotation>
  - <xsd:complexType>
    - <xsd:sequence>
      - <xsd:element ref="core:buyerLineItem" />
      - <xsd:element ref="dict:GlobalProductUnitOfMeasureCode" />
      - <xsd:element ref="dict:GlobalPurchaseOrderStatusCode" />
      - <xsd:element ref="core:isDropShip" />
      - <xsd:element ref="tns:OrderQuantity" />
      - <xsd:element ref="core:ProductIdentification" />
      - <xsd:element ref="core:requestedEvent" />
      - <xsd:element minOccurs="0" ref="core:requestUnitPrice" />
      - <xsd:element maxOccurs="unbounded" minOccurs="0" ref="core:ShippedQuantityInformation" />
      - <xsd:element minOccurs="0" ref="core:totalLineItemAmount" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
- <xsd:element name="OrderQuantity"/>

370
<xsd:complexType name="ReceiptAcknowledgmentType">
  <xsd:sequence>
    <xsd:element minOccurs="0" ref="tns:NonRepudiationlnformation" />
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="NonRepudiationlnformation">
  <xsd:annotation>
    <xsd:documentation>Information required to support non-repudiation of receipt.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:OriginalMessageDigest" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="OriginalMessageDigest">
  <xsd:annotation>
    <xsd:documentation>The base-64 encoded digest of the entire original mime message received. The digest MUST use the same algorithm as the original signed message.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType mixed="true">
    <xsd:choice maxOccurs="unbounded" minOccurs="0" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="ConfirmLineItemRequest" type="ms:LineItemConfirmationRequestType" />

<xsd:element name="ConfirmLineItemResponse" type="xsd:string" />

<xsd:element name="Pip3A4PurchaseOrderConfirmationSend">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="core:fromRole" />
      <xsd:element ref="tns:PurchaseOrder" />
      <xsd:element ref="core:requestingDocumentDateTime" />
      <xsd:element ref="core:requestingDocumentIdentifier" />
      <xsd:element ref="core:thisDocumentGenerationDateTime" />
      <xsd:element ref="core:thisDocumentIdentifier" />
      <xsd:element ref="core:toRole" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Pip3A4PurchaseOrderConfirmationSendResponse" type="tns:ReceiptAcknowledgmentType" />

<xsd:element name="SearchCustomerSend">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="searchString" nillable="false" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="SearchCustomerSendResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="customerId" nillable="false" type="xsd:long" />
      <xsd:element name="roleCode" nillable="false" type="xsd:string" />
      <xsd:element name="contactName" nillable="false" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

371
<xsd:element name="email" nillable="false" type="xsd:string" />
<xsd:element name="telephone" nillable="false" type="xsd:string" />
<xsd:element name="businessName" nillable="false" type="xsd:string" />
<xsd:element name="postalCode" nillable="false" type="xsd:string" />
<xsd:element name="city" nillable="false" type="xsd:string" />
<xsd:element name="street" nillable="false" type="xsd:string" />
<xsd:element name="countryCode" nillable="false" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CreateOrderRequestType">
<xsd:sequence>
<xsd:element name="authToken" nillable="false" type="xsd:string" />
<xsd:element name="contact" nillable="false" type="tns:ContactInfo" />
<xsd:element name="shipTo" nillable="false" type="tns:Address" />
<xsd:element minOccurs="0" name="billTo" nillable="false" type="tns:Address" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CreateOrderResponseType">
<xsd:sequence>
<xsd:element name="orderId" nillable="false" type="xsd:long" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Item">
<xsd:sequence>
<xsd:element name="articleId" nillable="false" type="xsd:string" />
<xsd:element name="quantity" nillable="false" type="xsd:int" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AddLineItemResponseType">
<xsd:sequence>
<xsd:element name="authToken" nillable="false" type="xsd:string" />
<xsd:element name="orderld" nillable="false" type="xsd:long" />
<xsd:element name="item" nillable="false" type="tns:Item" />
</xsd:sequence>
</xsd:complexType>

Appendix I

<xsd:complexType name="AddLineItemResponseType">
  <xsd:sequence>
    <xsd:element name="orderId" nillable="false" type="xsd:long" />
    <xsd:element name="lineItemld" nillable="false" type="xsd:long" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CloseOrderRequestType">
  <xsd:sequence>
    <xsd:element name="authToken" nillable="false" type="xsd:string" />
    <xsd:element name="orderId" nillable="false" type="xsd:long" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CloseOrderResponseType">
  <xsd:sequence>
    <xsd:element name="orderId" nillable="false" type="xsd:long" />
    <xsd:element name="itemsNo" nillable="false" type="xsd:int" />
  </xsd:sequence>
</xsd:complexType>

<xsd:schema>
  <wsdl:types>
    <xsd:complexType name="CreateOrderSendResponse">
      <xsd:sequence>
        <xsd:element name="CreateOrderRequest" nillable="false" type="schema:CreateOrderRequest" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>

<xsd:message name="CloseOrderSendResponse">
  <wsdl:part element="schema:CloseOrderSendResponse" name="CloseOrderSendResponse" />
</xsd:message>

<xsd:message name="AddLineItemSendResponse">
  <wsdl:part element="schema:AddLineItemSendResponse" name="AddLineItemSendResponse" />
</xsd:message>

<xsd:message name="CreateOrderSendResponse">
  <wsdl:part element="schema:CreateOrderSendResponse" name="CreateOrderSendResponse" />
</xsd:message>

<xsd:message name="ConfirmLineItemResponse">
  <wsdl:part element="schema:ConfirmLineItemResponse" name="ConfirmLineItemResponse" />
</xsd:message>

<xsd:message name="Pip3A4PurchaseOrderConfirmationSendResponse">
  <wsdl:part element="schema:Pip3A4PurchaseOrderConfirmationSendResponse" name="Pip3A4PurchaseOrderConfirmationSendResponse" />
</xsd:message>

<xsd:message name="Pip3A4PurchaseOrderRequest">
  <wsdl:part element="schema:Pip3A4PurchaseOrderRequest" name="Pip3A4PurchaseOrderRequest" />
</xsd:message>

<xsd:message name="Pip3A4PurchaseOrderResponse">
  <wsdl:part element="schema:Pip3A4PurchaseOrderResponse" name="Pip3A4PurchaseOrderResponse" />
</xsd:message>

<xsd:message name="SearchCustomerSendResponse">
  <wsdl:part element="schema:SearchCustomerSendResponse" name="SearchCustomerSendResponse" />
</xsd:message>

<xsd:message name="ConfirmLineItemRequest">
  <wsdl:part element="schema:ConfirmLineItemRequest" name="ConfirmLineItemRequest" />
</xsd:message>

<xsd:portType name="Mediator">
  <wsdl:operation name="Pip3A4PurchaseOrder">
    <wsdl:input message="tns:Pip3A4PurchaseOrderRequest" name="Pip3A4PurchaseOrderRequest" />
    <wsdl:output message="tns:Pip3A4PurchaseOrderResponse" name="Pip3A4PurchaseOrderResponse" />
  </wsdl:operation>
  <wsdl:operation name="ConfirmLineItem">
    <wsdl:input message="tns:ConfirmLineItemRequest" name="ConfirmLineItemRequest" />
    <wsdl:output message="tns:ConfirmLineItemResponse" name="ConfirmLineItemResponse" />
  </wsdl:operation>
</xsd:portType>

<xsd:binding name="MediatorBinding" type="tns:Mediator">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />  
</xsd:binding>

<xsd:operation name="Pip3A4PurchaseOrder">
  <soap:operation soapAction="" />
  <xsd:input name="Pip3A4PurchaseOrderRequest" />
</xsd:operation>
I.5 sBPEL for the Mediator Process

```xml
<soap:body use="literal" />
<wSDL:input>
<soap:body use="literal" />
<wSDL:output name="Pip3A4PurchaseOrderResponse">
<soap:body use="literal" />
</wSDL:output>
</wSDL:operation>
<wSDL:operation name="ConfirmLineItem">
<soap:operation soapAction="" />
<wsdl:input name="ConfirmLineItemRequest">
<soap:body use="literal" />
</wsdl:input>
<wSDL:output name="ConfirmLineItemResponse''">
<soap:bodv use="literal" />
</wsdl:output>
</wSDL:operation>
</wSDL:binding>
<wSDL:service name="MediatorService">
<wsdl:port binding="tns:MediatorBinding" name="MediatorPort''">
<soap:address location="http://localhost:8080/swsc/services" />
</wsdl:port>
</wSDL:service>
</wSDL:definitions>

1.5 sBPEL for the Mediator Process

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace _"http://ip-super.org/instances/auto-generated/sBPEL#"

sbpel _"http://ip-super.org/ontologies/process/sbpel/v2.0.0#",
bpel _"http://ip-super.org/ontologies/process/bpel20/v2.0.0#",
wsmostudio _"http://www.wsmostudio.org#" }

ontology MoonMediatorProcessInstance_sBPEL

nonFunctionalProperties

wsmostudio#version hasValue "0.7.3.2"
"http://purl.org/dc/elements/1.1#creator" hasValue "Automatically generated by BPMO2sBPEL Tool"
endNonFunctionalProperties

importsOntology

{""http://ip-super.org/ontologies/process/sbpel/v2.0.0#sbpel",
"http://ip-super.org/ontologies/process/bpel20/v2.0.0#bipel20",
"http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#MoonMediatorProcessInstance"}

instance Process_MoonMediator_sBPEL memberOf sbpel#SemanticProcess

bipel#hasName hasValue "Moon Mediator Process"
bipel#hasTargetNamespace hasValue "http://ip-super.org/instances/auto-generated/sBPEL"
bipel#hasPartner hasValue {moonOM_sBPEL, customer_sBPEL }
bipel#hasConversation hasValue {GoalTask_SearchCustomer_sBPELConversation,
GoalTask_CreateOrder_sBPELConversation, GoalTask_CloseOrder_sBPELConversation }
bipel#hasVariable hasValue {SemanticCapability_SearchCustomerDesc_sBPEL, SemanticCapability_OrderDesc_sBPEL,
SemanticCapability_ConvertLineDesc_sBPEL, SemanticCapability_POConfirmationDesc_sBPEL,
SemanticCapability_CloseOrderDesc_sBPEL, SemanticCapability_LineItemDesc_sBPEL,
SemanticCapability_OrderResponseDesc_sBPEL, SemanticCapability_PurchaseOrderDesc_sBPEL }
bipel#correspondsTo hasValue
"http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#Process_MoonMediator"
bipel#hasActivity hasValue StartEvent_1_sBPEL

instance customer_sBPEL memberOf sbpel#Partner
```
sbpel#hasName hasValue "Blue Customer"
sbpel#hasBusinessEntity hasValue "Blue, SWS Challenge"
sbpel#hasConversation hasValue Receive_ReceivPO_sBPELCOnversation

instance moonOM_sBPEL memberOf sbpel#Partner
sbpel#hasName hasValue "Moon Order Management"
sbpel#hasBusinessEntity hasValue "Moon, SWS Challenge"
sbpel#hasConversation hasValue Receive_ConfirmLineItemsBPELCOnversation

instance StartEvent_1_sBPEL memberOf bpel#Sequence
bpel#hasActivity hasValue ControlflowConnector_100_sBPEL

instance ControlflowConnector_100_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue Receive_ReceivPO_sBPEL
bpel#hasOrderedActivity hasValue ControlflowConnector_200_sBPEL

instance Receive_ReceivPO_sBPEL memberOf bpel#ExtensionActivity
bpel#correspondsTo hasValue "http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcesslnstance#Receive_ReceivePO"
bpel#hasActivity hasValue Receive_ReceivPO_sBPELReceive

instance SemanticCapability_PurchaseOrderDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_PurchaseOrderDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_PurchaseOrderDesc_sBPELWSDLMessageTyoe
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datanmediator#PurchaseOrder"

instance ControlflowConnector_200_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue MediationTask_MapPurchaseOrder_sBPEL
bpel#hasOrderedActivity hasValue ControlflowConnector_300_sBPEL

instance MediationTask_MapPurchaseOrder_sBPEL memberOf bpel#Assign
bpel#hasName hasValue "Map Purchase Order"
bpel#hasAssignOperation hasValue [DataMediatore_MapOrderRequestToSearchCustomer_sBPEL,
DataMediatore_MapOrderRequestToOrder_sBPEL ]
bpel#correspondsTo hasValue "http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcesslnstance#MediationTask_MapPurchaseOrder"

instance DataMediatore_MapOrderRequestToSearchCustomer_sBPEL memberOf sbpel#Mediate
sbpel#usesDataMediatore hasValue SemanticCapability_MapOrderRequestToSearchCustomer_sBPELDataMediatore
sbpel#hasInputVariable hasValue SemanticCapability_PurchaseOrderDesc_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_SearchCustomerDesc_sBPEL

instance DataMediatore_MapOrderRequestToOrder_sBPEL memberOf sbpel#Mediate
sbpel#usesDataMediatore hasValue SemanticCapability_MapOrderRequestToOrder_sBPELDataMediatore
sbpel#hasInputVariable hasValue SemanticCapability_PurchaseOrderDesc_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_OrderDesc_sBPEL

instance ControlflowConnector_300_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue GoalTask_SearchCustomer_sBPEL
bpel#hasOrderedActivity hasValue ControlflowConnector_400_sBPEL

instance GoalTask_SearchCustomer_sBPEL memberOf bpel#ExtensionActivity
bpel#correspondsTo hasValue "http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcesslnstance#GoalTask_SearchCustomer"
bpel#hasActivity hasValue GoalTask_SearchCustomer_sBPELSendReceive

instance SemanticCapability_SearchCustomerDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_SearchCustomerDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_SearchCustomerDesc_sBPELWSDLMessageTyoe
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datanmediator#SearchCustomer"

instance SemanticCapability_SearchCustomerResponseDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_SearchCustomerResponseDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_SearchCustomerResponseDesc_sBPELWSDLMessageTyoe
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datanmediator#SearchCustomerResponse"

instance ControlflowConnector_400_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue GoalTask_CreateOrder_sBPEL
bpel#hasOrderedActivity hasValue ControlflowConnector_500_sBPEL

375
instance GoalTask_CreateOrder_sBPEL memberOf bpel#ExtensionActivity
bpel#hasCorrespondsTo hasValue _"http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#GoalTask_CreateOrder"
bpel#hasActivity hasValue GoalTask_CreateOrder_sBPEL:SendReceive

instance SemanticCapability_OrderDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_OrderDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_OrderDesc_sBPEL:WSDLMessage_Type
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datamediator#Order"

instance SemanticCapability_OrderResponseDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_OrderResponseDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_OrderResponseDesc_sBPEL:WSDLMessage_Type
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datamediator#NewOrderResponse"

instance ControlflowConnector_500_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue Repeat_AddLineItem_sBPEL
bpel#hasOrderedActivity hasValue ControlflowConnector_600_sBPEL

instance Repeat_AddLineItem_sBPEL memberOf bpel#RepeatUntil
bpel#hasCondition hasValue Condition_1218020116299_1731096821_sBPEL
bpel#hasActivity hasValue Send_AddLineItem_sBPEL

instance Condition_1218020116299_1731096821_sBPEL memberOf bpel#Condition
bpel#hasExpression hasValue "Another Item?"

instance Send_AddLineItem_sBPEL memberOf bpel#ExtensionActivity
bpel#hasCorrespondsTo hasValue _"http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#Send_AddLineItem"
bpel#hasActivity hasValue Send_AddLineItem_sBPEL:Send

instance SemanticCapability_LineItemDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_LineItemDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_LineItemDesc_sBPEL:WSDLMessage_Type
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datamediator#LineItem"

instance ControlflowConnector_600_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue GoalTask_CloseOrder_sBPEL
bpel#hasOrderedActivity hasValue ControlflowConnector_700_sBPEL

instance GoalTask_CloseOrder_sBPEL memberOf bpel#ExtensionActivity
bpel#hasCorrespondsTo hasValue _"http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#GoalTask_CloseOrder"
bpel#hasActivity hasValue GoalTask_CloseOrder_sBPEL:SendReceive

instance SemanticCapability_CloseOrderDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_CloseOrderDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_CloseOrderDesc_sBPEL:WSDLMessage_Type
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datamediator#CloseOrder"

instance SemanticCapability_CloseOrderResponseDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_CloseOrderResponseDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_CloseOrderResponseDesc_sBPEL:WSDLMessage_Type
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datamediator#CloseOrderResponse"

instance ControlflowConnector_700_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue Repeat_ConfirmLineItem_sBPEL
bpel#hasOrderedActivity hasValue ControlflowConnector_800_sBPEL

instance Repeat_ConfirmLineItem_sBPEL memberOf bpel#RepeatUntil
bpel#hasCondition hasValue Condition_1218020116299_2182159300_sBPEL
bpel#hasActivity hasValue Receive_ConfirmLineItem_sBPEL

instance Condition_1218020116299_2182159300_sBPEL memberOf bpel#Condition
bpel#hasExpression hasValue "Another Item?"

instance Receive_ConfirmLineItem_sBPEL memberOf bpel#ExtensionActivity
bpel#hasCorrespondsTo hasValue _"http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#Receive_ConfirmLineItem"
bpel#hasActivity hasValue Receive_ConfirmLineItem_sBPEL:Receive
instance SemanticCapability_ConfirmLineItemDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_ConfirmLineItemDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_ConfirmLineItemDesc_sBPELWSDLMessageType
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datamediator#OrderLineItemConfirmation"

instance ControlFlowConnector_800_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue MediationTask_MapResult_sBPEL
bpel#hasOrderedActivity hasValue ControlFlowConnector_900_sBPEL

instance MediationTask_MapResult_sBPEL memberOf bpel#Assign
bpel#hasName hasValue "Map Result"
bpel#hasAssignOperation hasValue DataMediator_MapRequestToConfirmation_sBPEL
bpel#correspondsTo hasValue "http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#MediationTask_MapResult"

instance DataMediator_MapRequestToConfirmation_sBPEL memberOf bpel#mediate
sbpel#usesDataMediator hasValue SemanticCapability_MapRequestToConfirmationDesc_sBPEL
sbpel#hasInputVariable hasValue SemanticCapability_PurchaseOrderDescs_sBPEL
sbpel#hasOutputVariable hasValue SemanticCapability_POConfirmationDescs_sBPEL

instance ControlFlowConnector_900_sBPEL memberOf bpel#OrderedActivity
bpel#hasActivity hasValue SendSendPOConf_sBPEL

instance SendSendPOConf_sBPEL memberOf bpel#ExtensionActivity
bpel#correspondsTo hasValue "http://lilianathesis.org/usecase/swsc/bpmo/MoonMediatorProcessInstance#Send_SendPOConf"
bpel#hasActivity hasValue SendSendPOConf_sBPELSend

instance SemanticCapability_POConfirmationDesc_sBPEL memberOf sbpel#SemanticVariable
bpel#hasName hasValue "SemanticCapability_POConfirmationDesc_sBPEL"
bpel#hasType hasValue SemanticCapability_POConfirmationDesc_sBPELWSDLMessageType
sbpel#hasSemanticType hasValue "http://lilianathesis.org/usecase/swsc/datamediator#PurchaseOrderConfirmation"

instance GoalTask_SearchCustomer_sBPELSendReceive memberOf sbpel#SendReceive
bpel#hasName hasValue "Search Customer"
bpel#hasInputVariable hasValue SemanticCapability_SearchCustomerDesc_sBPEL
bpel#hasOutputVariable hasValue SemanticCapability_SearchCustomerResponseDesc_sBPEL
sbpel#belongsToConversation hasValue GoalTask_SearchCustomer_sBPELConversation

instance GoalTask_CreateOrder_sBPELSendReceive memberOf sbpel#SendReceive
bpel#hasName hasValue "Create Order"
bpel#hasInputVariable hasValue SemanticCapability_OrderDesc_sBPEL
bpel#hasOutputVariable hasValue SemanticCapability_OrderResponseDesc_sBPEL
sbpel#belongsToConversation hasValue GoalTask_CreateOrder_sBPELConversation

instance Send_AddLineItem_sBPELSend memberOf sbpel#Send
bpel#hasName hasValue "Add Line Item"
bpel#hasVariable hasValue SemanticCapability_LineItemDesc_sBPEL
sbpel#belongsToConversation hasValue Receive_ConfirmLineItem_sBPELConversation

instance GoalTask_CloseOrder_sBPELSendReceive memberOf sbpel#SendReceive
bpel#hasName hasValue "Close Order"
bpel#hasInputVariable hasValue SemanticCapability_CloseOrderDesc_sBPEL
bpel#hasOutputVariable hasValue SemanticCapability_CloseOrderResponseDesc_sBPEL
sbpel#belongsToConversation hasValue GoalTask_CloseOrder_sBPELConversation

instance Receive_ConfirmLineItem_sBPELSendReceive memberOf sbpel#Receive
bpel#hasName hasValue "Confirm Line Item"
bpel#hasVariable hasValue SemanticCapability_ConfirmLineItemDesc_sBPEL
sbpel#belongsToConversation hasValue Receive_ConfirmLineItem_sBPELConversation

instance Send_SendPOConf_sBPELSend memberOf sbpel#Send
bpel#hasName hasValue "Send PO Confirmation"
bpel#hasVariable hasValue SemanticCapability_POConfirmationDesc_sBPEL
sbpel#belongsToConversation hasValue ReceiveReceivePO_sBPELConversation

instance ReceiveReceivePO_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "ReceiveReceivePO_sBPELConversation"
sbpel#describesInterface hasValue ReceiveReceivePO_sBPELInterface

instance GoalTask_SearchCustomer_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "GoalTask_SearchCustomer_sBPELConversation"
sbpel#describesInterface hasValue GoalTask_SearchCustomer_sBPELIncomingInterface

instance GoalTask_CreateOrder_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "GoalTask_CreateOrder_sBPELConversation"
sbpel#describesInterface hasValue GoalTask_CreateOrder_sBPELOutgoingInterface

instance GoalTask_CloseOrder_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "GoalTask_CloseOrder_sBPELConversation"
sbpel#describesInterface hasValue GoalTask_CloseOrder_sBPELOutgoingInterface

instance Receive_ConfirmLineitem_sBPELConversation memberOf sbpel#Conversation
sbpel#hasName hasValue "Receive_ConfirmLineitem_sBPELConversation"
sbpel#describesInterface hasValue Receive_ConfirmLineitem_sBPELInterface

instance ReceiveReceivePO_sBPELInterface memberOf sbpel#IncomingInterface
sbpel#hasWebServiceDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/RequestPOWS#RequestPOWS"

instance GoalTask_SearchCustomer_sBPELOutgoingInterface memberOf sbpel#OutgoingInterface
sbpel#hasGoalDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/SearchCustomerGoal#SearchCustomerGoal"

instance GoalTask_CreateOrder_sBPELOutgoingInterface memberOf sbpel#OutgoingInterface
sbpel#hasGoalDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/CreateOrderGoal#CreateOrderGoal"

instance GoalTask_CloseOrder_sBPELOutgoingInterface memberOf sbpel#OutgoingInterface
sbpel#hasGoalDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/CloseOrderGoal#CloseOrderGoal"

instance Receive_ConfirmLineitem_sBPELInterface memberOf sbpel#IncomingInterface
sbpel#hasWebServiceDescription hasValue "http://lilianathesis.org/usecase/swsc/wsmo/ConfirmLineitemWS#ConfirmLineitemWS"

instance SemanticCapability_MapOrderRequestToSearchCustomer_sBPELDataMediator memberOf sbpel#DataMediator
sbpel#usesMediator hasValue "http://lilianathesis.org/usecase/swsc/datamediator#SearchCustomerFromPurchaseOrderRequest"

instance SemanticCapability_MapOrderRequestToOrder_sBPELDataMediator memberOf sbpel#DataMediator
sbpel#usesMediator hasValue "http://lilianathesis.org/usecase/swsc/datamediator#OrderFromPurchaseOrderRequest"

instance SemanticCapability_MapRequestToConfirmation_sBPELDataMediator memberOf sbpel#DataMediator
sbpel#usesMediator hasValue "http://lilianathesis.org/usecase/swsc/datamediator#ConfirmationFromPurchaseOrderRequest"

instance SemanticCapability_PurchaseOrderDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_SearchCustomerDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_SearchCustomerResponseDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_OrderDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_OrderResponseDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_LineitemDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""

instance SemanticCapability_CloseOrderDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType
bpel#hasDefinition hasValue ""
instance SemanticCapability_CloseOrderResponseDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType bpel#hasDefmition hasValue ""

instance SemanticCapability_ConfirmLineItemDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType bpel#hasDefmition hasValue ""

instance SemanticCapability_POConfirmationDesc_sBPELWSDLMessageType memberOf bpel#WSDLMessageType bpel#hasDefmition hasValue ""