



Data, Information and Process Integration
with Semantic Web Services

DIP

Data, Information and Process Integration with Semantic Web Services

FP6 - 507483

Deliverable

WP 8: Case Study B2B in Telecommunications

D8.3

Prototype Platform Design

Sam Watkins

Sinuhé Arroyo

Marc Richardson

Bernhard Schreder

Alex Wahler

July 1st, 2005



Executive Summary

This deliverable presents the design of a Business-to-Business platform for the telecommunications industry based upon Semantic Web Services. The aims of such a platform are to enable telecommunications companies to operate more effectively in what is an increasingly dynamic, multi-disciplinary business environment; requiring the delivery of innovative services to customers more quickly and with reduced cost. The objective of the deliverable is to provide the platform design in terms of an outline architecture and model scenarios in WSMO.

The specific use case - Assurance Integration, focusing on improving the integration process for an existing B2B Operational Support Systems (OSS) Gateway - is presented. WSMO descriptions for the use case are provided along with associated ontologies. The intention is to show how semantically described interfaces can reduce the time and costs involved in integrating the heterogeneous OSS systems of partner companies in order to establish workflow processes along the supply chain. The Assurance Integration use case was presented in detail in DIP Deliverable 8.2 [4]. The current platform, BT Wholesale's B2B Gateway known as eCo, uses ebXML to format requests from and responses to partner company systems. This causes issues when new trading partners try to access eCo functionality, requiring a point-to-point integration exercise for each new partner. In summary, this use case shows the eCo Platform can be enhanced by using semantic descriptions in the interfaces to its customers, who are service resellers and need to be able to communicate with the wholesaler's systems, for example to raise fault reports and requests for tests.

The prototype uses a set of DIP technology components. These are described together with a selection process to determine the configuration of components in the use case prototype system. The prototype provides the means to mediate between the data and process representations used by eCo and the partners' systems at the semantic level and to apply those mediators to messages as they arrive (or leave) the platform. Thus the high level functionality required is:

1. Semantic representation of interfaces (both in terms of data and process) using WSMO
2. Creation of data mediators
3. Creation of process mediators
4. Adaptation of native messages to WSML format (lifting)
5. Adaptation of WSML messages to native format (lowering)
6. Data mediation of WSML messages
7. Process mediation of WSML messages
8. Invocation of service/interface

Items 1-3 are design-time activities. These could either be enabled by authoring directly in WSML or by using an appropriate GUI if available.

Items 4-8 are run-time activities which will make use of the output of activities 1-3 to allow valid interaction to occur. At a high level, the sequence of events at runtime can be described as:

- A service interface sends a message in its native format (e.g. XML)
- Data and process mediation is applied to the message to provide interoperability with the target platform (here eCo)
- The response from the eCo platform is similarly mediated and a message sent back to the service originating the interaction in its native format

Thus any trading partner using any message format will be able to access the eCo platform. Also, by using the adapters, the interface to the back end eCo platform is maintained by using the WSML interface. While this is intended to run with the ebXML messages in the existing platform, the eventual adoption of a SWS approach would remove the need for such an interface. In this scenario, an execution environment such as WSMX would directly call the back-end systems.

The set of components required includes a set of design-time tools to create the semantic descriptions of the services and mediators, the WSMX core, communication manager and resources manager to manage storage and interaction, the adaptor framework to convert to and from WSML and mediator components to apply the WSMO mediators at run-time.

A WSMX process mediation component is not currently available and therefore will not initially form part of the prototype. The SOPHIE [3] choreography service, which has been developed within Workpackage 8, provides an initial approach to allow the representation of choreography at design time and subsequent process mediation at run-time. The lessons learned in the development and usage of SOPHIE will be used to provide feedback about the requirements needed for process mediation to the corresponding DIP workpackage (WP5). When WSMX choreography and process mediation become available they will be applied to the use case.

For the Assurance Integration use case, we have modelled the behaviour of the eCo services with the aim of allowing easier interoperation with heterogeneous partner systems. We have also examined and discussed which technical components will be available in the time frame of (and hence used in) the production of the prototype Deliverable 8.4.

The architecture proposed in this deliverable allows the description and use of SWS that will form an integral part of a Semantically Enabled Service Orientated Architecture, demonstrating this as a viable approach to producing an extensible prototype in response to the requirements presented in Deliverable 8.2.

Future work will evaluate the prototype with respect to technical and non-technical requirements. One key evaluation exercise will involve BT's B2B enablement team who are effectively the customers of this work and one or more of BT Wholesale's trading partners. The intention is to evaluate the applicability and benefits of the approach insitu.

The prototype described in this deliverable deals with a subset of expected benefits of a SWS approach which can be illustrated given the current availability of DIP's technology components. A second prototype is planned in DIP which will extend the work presented here to include automated design and run-time discovery and composition. This will be based around BT's ICT Platform (currently under development and based upon traditional Web Services) which will exist as a (manual) discovery and composition facility for BT's Service Orientated Architecture. The

platform will allow BT and 3rd party services and capabilities to be combined to form customer facing products. The intention of the DIP prototype is to show how greater automation can be achieved with the adoption of SWS giving customers the ability to create service bundles on demand. The second prototype will be preceded by a design deliverable in the same manner as this one.

As the prototype design for Assurance Integration using Semantic Web Services, this deliverable contributes to one of DIP's main goals: "***Apply Semantic Web Services as an infrastructure in real world scenarios within an organization and between organizations and its customers, partners and suppliers***". The core component for this prototype is the open source architecture WSMX. The prototype provides a proof of concept of the WSMX platform in the area of B2B integration in the telecommunications sector.

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

Document Information

IST Project Number	FP6 – 507483	Acronym	DIP
Full title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	http://dip.semanticweb.org		
Document URL			
EU Project officer	Kai Tulus		

Deliverable	Number	8.3	Title	Prototype Platform Design
Work package	Number	8	Title	B2B Telecommunications Integration


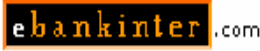



Date of delivery	Contractual	M 17	Actual	
Status	Version. 0.1		final	<input checked="" type="checkbox"/>
Nature	Prototype <input type="checkbox"/> Report <input checked="" type="checkbox"/> Dissemination <input type="checkbox"/>			
Dissemination Level	Public <input type="checkbox"/> Consortium <input checked="" type="checkbox"/>			

Authors (Partner)	Sinuhé Arroyo (DERI), Bernhard Schreder (NIWA), Marc Richardson (BT), Alex Wahler (NIWA), Sam J Watkins (BT)			
Responsible Author	Sam J Watkins		Email	Sam.Watkins@bt.com
	Partner	BT	Phone	+44 (0) 1473 609636
Author	Marc Richardson		Email	marc.richardson@bt.com
	Partner	BT	Phone	+44 (0) 1473 609589
Author	Bernhard Schreder		Email	Schreder@niwa.at
	Partner	NIWA	Phone	+31 (0) 1 3195843 – 53
Author	Alexander Wahler		Email	wahler@niwa.at
	Partner	NIWA	Phone	+31 (0) 1 3195843 – 11
Author	Sinuhé Arroyo		Email	sinuhe.arroyo@deri.org
	Partner	DERI	Phone	+43 (0) 512 507 6489




<p>Abstract (for dissemination)</p>	<p>This deliverable presents the core design for the prototype that will be developed in DIP deliverable 8.4.</p> <p>In those terms, the WSMO message descriptions are presented, along with requirements such a prototype has on the DIP platform.</p> <p>Hence, one may see how the DIP Architecture can be usefully deployed to ease the integration issues that exist in Telecommunications where B2B interaction is increasingly important.</p> <p>It performs this by focusing on the “Assurance Integration” use case presented in DIP D8.2, which discusses the problem of how to offer assurance tests for supplier services, guaranteeing a specific quality of service in the network for their third party services by establishing a workflow throughout the supply chain and ensuring that the appropriate parties can resolve issues as necessary.</p>	
<p>Keywords</p>	<p>Telecommunications B2B Integration Assurance Integration WSMO descriptions, WSMX</p>	

Version Log			
Issue Date	Rev No.	Author	Change
01-07-05	Version 1	Sam J Watkins et al.	Released

Project Consortium Information

Partner	Acronym	Contact
National University of Ireland Galway	NUIG  <i>Ollscoil na hÉireann, Gaillimh</i>	Prof. Dr. Christoph Bussler Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland Email: chris.bussler@deri.org Tel: +353 91 512460
Fundacion De La Innovacion.Bankinter	Bankinter 	Monica Martinez Montes Fundacion de la Innovacion. BankInter Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
Berlecon Research GmbH	Berlecon 	Dr. Thorsten Wichmann Berlecon Research GmbH Oranienburger Str. 32 10117 Berlin, Germany Email: tw@berlecon.de Tel: +49 30 2852960
British Telecommunications Plc.	BT 	Dr John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham Ipswich IP5 3RE, United Kingdom Email: john.nj.davies@bt.com Tel: +44 1473 609583
Swiss Federal Institute of Technology, Lausanne	EPFL 	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Fédérale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland Email : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Essex County Council	Essex 	Mary Rowlett, Essex County Council PO Box 11, County Hall, Duke Street Chelmsford, Essex, CM1 1LX United Kingdom. Email: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI 	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany Email: abecker@fzi.de Tel: +49 721 9654 0

Partner	Acronym	Contact
Institut für Informatik, Leopold-Franzens Universität Innsbruck	UIBK 	Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: dieter.fensel@deri.org Tel: +43 512 5076485
ILOG SA	ILOG 	Christian de Sainte Marie 9 Rue de Verdun, 94253 Gentilly, France Email: csma@ilog.fr Tel: +33 1 49082981
inubit AG	Inubit 	Torsten Schmale inubit AG Lützowstraße 105-106 D-10785 Berlin Germany Email: ts@inubit.com Tel: +49 30726112 0
Intelligent Software Components, S.A.	iSOCO 	Dr. V. Richard Benjamins, Director R&D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain Email: rbenjamins@isoco.com Tel. +34 913 349 797
NIWA WEB Solutions	NIWA 	Alexander Wahler NIWA WEB Solutions Niederacher & Wahler OEG Kirchengasse 13/1a A-1070 Wien Email: wahler@niwa.at Tel:+43(0)1 3195843-11
The Open University	OU 	Dr. John Domingue Knowledge Media Institute The Open University, Walton Hall Milton Keynes, MK7 6AA United Kingdom Email: j.b.domingue@open.ac.uk Tel.: +44 1908 655014
SAP AG	SAP 	Dr. Elmar Dorner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany Email: elmar.dorner@sap.com Tel: +49 721 6902 31

<p>Sirma AI Ltd.</p>	<p>Sirma</p>  <p>Ontotext Knowledge and Language Engineering Lab of Sirma</p>	<p>Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse Sofia 1784, Bulgaria Email: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303</p>
<p>Unicorn Solution Ltd.</p>	<p>Unicorn</p> 	<p>Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951 Israel Email: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111</p>
<p>Vrije Universiteit Brussel</p>	<p>VUB</p>  <p>Vrije Universiteit Brussel</p>	<p>Pieter De Leenheer Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel ,Belgium Email: Pieter.De.Leenheer@vub.ac.be Tel.: +32 (0) 2 629 3749</p>

List OF Key Words/Abbreviations

API	Application Programming Interface
B2B	Business to Business
CCBS	Customer Care and Billing System
CRM	Customer Relationship Management
ebXML	Electronic Business using eXtensible Mark-up Language examined in D8.2 [4]
eCo	BT Wholesale B2B OSS Gateway, run by BT Wholesale.
eTOM	Enhanced Telecom Operations Map
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
MVC	Model View Controller – an application framework which should allow easily configurable front-ends to be produced, with stable business logic attributes (models) but instable UI's (views)
OSS	Operational Support System
SOAP	Simple Object Access Protocol
SOPHIE	Semantic chOreograPHy servIce
SWS	Semantic Web Service
SWWS	Semantic Web enabled Web Services
UBL	Universal Business Language
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator
WSML	Web Service Modelling Language
WSMO	Web Services Modelling Ontology
XML	Extensible Mark-up Language
XSLT	Extensible Style Language Transformation

Table of Contents

EXECUTIVE SUMMARY	I
LIST OF KEY WORDS/ABBREVIATIONS	VI
TABLE OF CONTENTS	X
1 INTRODUCTION	13
1.1 Design Guidelines	14
2 OVERVIEW OF ASSURANCE INTEGRATION USE CASE	16
2.1 Actors involved	16
2.2 Test Request	17
2.2.1 The Test Request process	17
2.2.2 Scenario description for the Test Request process	18
2.3 Overview of the other functions performed by the eCo Platform	19
2.4 Message Conversion.....	20
2.4.1 Scenario description	20
2.4.2 Sequence diagram.....	20
2.5 Ontology Mapping and Data Mediation.....	21
2.5.1 Use case diagram	21
2.5.2 Scenario description	21
2.5.3 Sequence diagram.....	22
3 ARCHITECTURE OF THE ASSURANCE INTEGRATION PROTOTYPE	23
4 ASSURANCE INTEGRATION DESIGN	26
4.1 Design of interface components	26
4.1.1 Web Services	26
4.1.2 UI and Front-ends.....	26
4.1.3 Adapters within the Assurance Integration Prototype.....	27
4.2 Services.....	30
4.2.1 Domain Ontology	30
4.2.2 WSMO Descriptions	35
4.2.2.1 Goals.....	35
4.2.2.2 Web Services	36
4.2.2.3 Mediators	39
4.2.2.4 Choreography - SOPHIE	40
4.2.2.5 Choreography Example	41
5 IMPLEMENTATION OF THE PROTOTYPE	46
5.1 Assurance Integration Test Request	46
5.2 Message Conversion.....	47

5.3 Integration using the DIP API	48
5.4 SOPHIE Implementation details	49
5.4.1 Topologies	49
5.4.2 Overall Algorithm	50
5.4.3 Integration with the prototype	51
6 CONCLUSIONS AND FUTURE WORK.....	52
6.1 Conclusions from this deliverable	52
6.2 Future Work.....	52
7 REFERENCES	53
APPENDIX 1: WSMO DESCRIPTIONS	54
The Domain Ontology	54
The WSMML Descriptions for the defined services	60
RequestTest	60
RequestTroubleTicket	64
RequestAppointmentAvailability.....	68
MakeAppointment.....	71
RequestAppointmentCancellation.....	74
APPENDIX 2: CONCEPTUAL CHOREOGRAPHY ONTOLOGY	78

LIST OF FIGURES

Figure 1: Assurance Integration Use Case 17

Figure 2: B2B Assurance Integration sequence diagram for any test request..... 18

Figure 3 Convert Message sequence diagram 20

Figure 4: Data Mediation Use Case..... 21

Figure 5: Data Mediation sequence diagram..... 22

Figure 6: WSMX being used in the Telecommunications B2B Integration Platform.... 24

Figure 7: Message Flow and Adapters Required in D8.4 27

Figure 8: Overview of lifting and lowering XML messages into WSML messages 28

Figure 9: Adapter Framework 29

Figure 10: Eco Ontology 32

Figure 11 - Enlarged Eco Ontology (part 1)..... 33

Figure 12 -Enlarged Eco Ontology (part 2)..... 34

Figure 13: In-Multi-Out Message Exchange Pattern..... 42

Figure 14: Request-Response and In-Multi-Out Message Exchange Pattern 43

Figure 15: Merge Box for testReply message 45

Figure 16: Test Request Sequence Diagram..... 47

Figure 17: Convert Message (XML2WSML) sequence diagram 48

Figure 18: Hub and spoke topology 50

LIST OF TABLES

Table 1 Inputs to *TestRequest* transaction 37

Table 2 Structural model of SOPHIE 40

Table 3 Behavioural model of SOPHIE 41

Table 4. Example of the domain ontologies λ and λ' 44

Table 5. Example of the choreography ontologies θ , θ' 44

Table 6. Inputs to *TestRequest* transaction 49

Table 7. Pseudocode of the *run* algorithm..... 50

1 Introduction

This deliverable presents the design of the first prototype for the B2B in Telecommunications case study. The prototype focuses on the Assurance Integration use case described in DIP Deliverable 8.2. The prototype forms the first step in the development of a Business-to-Business platform for the Telecommunications industry based upon Semantic Web Services. The aims of such a platform are to enable Telecommunications companies to operate more effectively in what is an increasingly dynamic, multi-disciplinary business environment; delivering innovative services to customers more quickly and with reduced cost. The objective of the deliverable is to provide the outline architecture and design for the Assurance Integration prototype that will be delivered at M18 (June 2005). The prototype will make use of components that have been developed within DIP and the WSMO/WSMX working groups with the intention of demonstrating the benefits of semantic-based integration. A second, more comprehensive, prototype focussed on BT's ICT Service Platform is due for delivery in M30 (June 2006). This will be preceded by a design deliverable which will describe how the first prototype is to be extended.

The specific use case of Assurance Integration, focusing on improving the integration process for an existing B2B Operational Support Systems (OSS) Gateway (presented in D8.2) is summarised. WSMO descriptions and message exchanges for the use case are provided along with associated ontologies. The intention of the prototype is to show how semantically described interfaces can reduce the time and costs involved in integrating the heterogeneous OSS systems of partner companies in order to establish a workflow throughout the supply chain. This is a good example of a more general challenge facing the telecommunications industry. This prototype will address that goal by showing the benefits gained from describing and using the SWS that will form an integral part of a Semantically enabled Service Orientated Architecture.

The prototype produced in D8.4 will be based on the DIP Architecture as described in DIP deliverables D6.2 and D6.5, and D6.8. Indeed, much of the prototype will use the components produced by the tool work packages, DIP Work Packages 2, 3, 4 and 5. As such, this deliverable does not need to reproduce the work from these work packages but can use these results confidently. In turn these work packages use the results from the WSMX project. The prototype makes use of components developed in the DIP technology workpackages (1-5) and the associated WSMO working groups.

The deliverable is structured as follows: in Section 2, we provide an overview of the Assurance Integration Prototype in reference to the use case scenarios.

Section 3 provides a high-level prototype architecture where components described in section 2 are selected based upon the requirements of the use case.

In Section 4 (Assurance Integration Design) we show the design meeting the requirements presented in DIP 8.2, including WSMO descriptions, ontologies used, front ends required and all mediation and conversion techniques to be deployed in the prototype.

Section 5 shows the planned Implementation of the Prototype, demonstrating the steps being taken to produce the prototype with the tools discussed in the architecture and design sections.

We conclude with a summary of work done and close with the aims of the prototypes in DIP Deliverable 8.5.

The Appendices show WSMO descriptions for the prototype and a Conceptual Choreography Ontology for greater depth and background reading.

1.1 Design Guidelines

This section provides an initial set of design guidelines for building a B2B Integration platform using the DIP Architecture and DIP tools. As a proof of concept these guidelines are applied to build the first B2B integration prototype and will be enhanced in the course of the project following the maturity of the tools. The basic prototype architecture should be built in a way that makes its adaptation to a specific application scenario (such as the one mentioned in section 3) straightforward.

Therefore this section provides a kind of design “recipe”, highlighting the main areas of involvement with the DIP Architecture and the custom components that have to be built on top of it to allow for the integration of front-end web applications and (Semantic) Web Services.

We distinguish between three major phases that need to be run through:

- Preparation Phase
- Design Phase
- Implementation Phase

Preparation Phase: The preparation phase starts with an analysis of the problem that has to be solved. We propose to prepare a narrative of the problem and match the demanded functionalities with those of the B2B integration platform. The next steps are descriptions of existing systems, a specification of the interfaces and a definition of actors and processes between these actors. State of the art software engineering tools and description languages as UML should be used. In detail the preparation phase includes the following steps:

- Analyse Problem
- Provide a narrative of the problem
- Match functionalities with B2B integration platform criteria (see D8.2)
- Define actors and roles (using techniques such as use case scenario descriptions and appropriate diagrams)
- Define major processes and interactions between the different actors. (e.g. by using sequence diagrams)
- Specify interfaces with existing components and define possible message exchanges (API, Message Exchange Patterns).
- Provide an overview of the system architecture, with a special focus on the integration of the B2B platform with existing (legacy) components.

In this use case, the Preparation Phase has been covered in D8.2 [4] and will be covered in Section 2 of this deliverable.

Design Phase: The design phase covers 6 major steps that are necessary to build the B2B integration platform. How these guidelines are applied to a real world scenario is described in section 4.

1. Ontologies have to be developed for both parties involved in the communication. These will be low level ontologies representing messages and a domain ontology which will give context to the elements of the messages.
2. Semantic service descriptions have to be produced for the back-end web services and published to a semantic repository accessible to the DIP Architecture.
3. Data mediation is required for the messages where different ontologies are being used.
4. Messages have to be defined and suitable Message Exchange Patterns have to be selected, both for requests and the services. These use a choreography ontology, also defined at design time.
5. Adapter components for message conversion have to be designed according to specific guidelines, as put forth by WP6.
6. A suitable front-end has to be designed, which will submit suitable messages to the B2B Integration platform. The messages have to commit to the format defined in step 4.

For this use case, Sections 3 and 4 of this deliverable address this phase.

Implementation Phase: The implementation phase covers the development of the components designed in the previous phase, the integration with existing systems and the deployment in the chosen infrastructure. Examples are listed below:

- The adapter components have to be implemented and deployed as web services on a suitable machine.

The front-end will may be any suitable web application, including one implemented as a Java/XML based web application or a simple Java Swing application.

For this use case, section 5 of this deliverable addresses this phase, along with deliverable D8.4.

2 Overview of Assurance Integration Use Case

This section provides a high level overview of the Assurance Integration Use Case. It examines the actors involved, the test request sequence (which is an interface presented by the eCO platform and the one used for illustration purposes throughout this deliverable), message conversion, the ontology mapping and management of the ontology in practice.

An overview of the eCo is also given in this section.

2.1 Actors involved

This section details the actors participating in the Assurance Integration scenario. While some of the actors were first mentioned in the description of the use case in deliverable D8.2, these paragraphs expand upon the information found there, incorporating all actors to be found in a typical scenario:

- The Customer is a customer of the Service Provider, from whom he has bought products for which BT Wholesale has provided services. The Customer has the possibility to raise an error with its Service Provider, should a problem with one of its products occur. Errors can be raised by using a Service Provider's online ticketing system.
- The Service Provider is a customer of BT Wholesale. These actors are any service provider or reseller with customers. The products they offer are in part dependent on services which they have bought from BT Wholesale. They provide online user interfaces for their customers – a ticketing system – to allow for customer support in the case of malfunctioning equipment or faulty services.
- BT Wholesale is a provider of wholesale telecoms equipment and services to the Service Provider. BT Wholesale provides web services which can be used to act upon test requests from Service Providers. Should a Customer raise an error with its Service Provider, the provider can request certain tests to be made by BT Wholesale, in order to determine whether or not it is their equipment or services that are responsible for the error.
- B2B Telecommunication Integration Platform, which is based on the DIP Architecture. For this first prototype, this means the use of the WSMX implementation of the DIP architecture provided by WP6 and all additionally necessary components. These components include at least Process Mediation and Choreography, Data Mediation, Invocation. In addition several custom adapter components have been added to the DIP Architecture, to allow for message conversions.

In the specific use case, multiple Service Providers are interfacing with one Wholesale Provider (BT). As such, the diversity is at one end only. However, outside of the use case, the scenario could easily be expanded to include diversity at both ends where Service Providers interact with multiple Wholesale Providers (depending on location, technology, (e.g. ADSL, cable, satellite) etc.). In this scenario, extra benefit is provided to the Service Providers as they are able to more easily reconfigure their systems and processes to interact with alternative Wholesale Providers. The benefit for BT Wholesale is not enhanced directly in this extended scenario which is why the use case

initially examines the scenario where diversity is at one end. However, as partners can use BT’s systems more easily, trust is more easily established and enhanced business relations may follow. In turn, BT does not have to expend time on integration problems, saving time in releasing products to a wider audience.

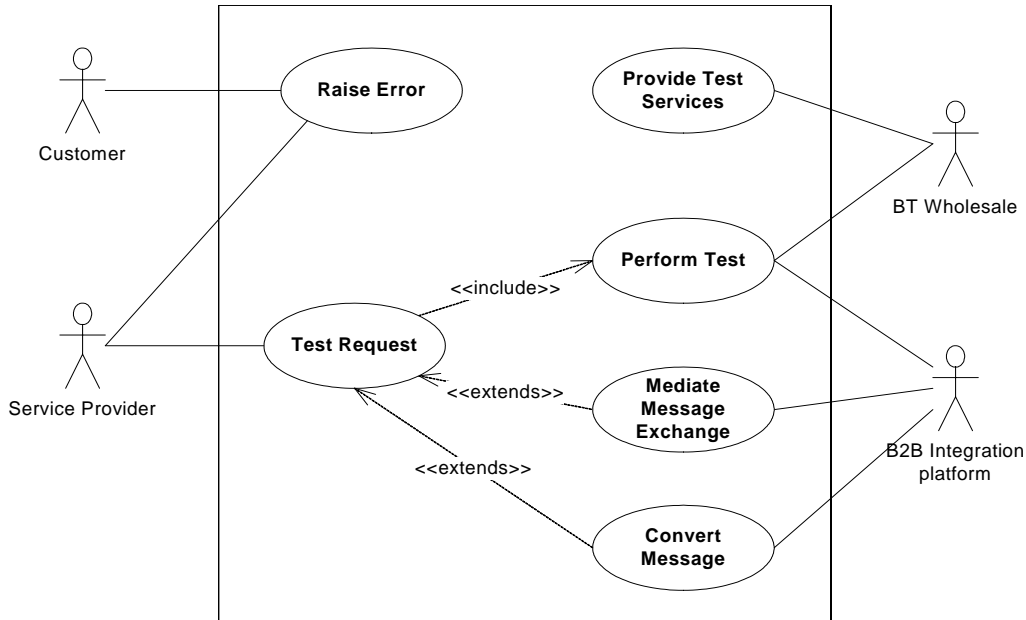


Figure 1: Assurance Integration Use Case

2.2 Test Request

The Test Request is one functional part of the first B2B Integration platform prototype and is the one used for illustration purposes in this deliverable. It simulates the activities necessary to undertake after the raising of an error by a Service Provider’s Customer. The exact procedures involved in the current realisation of this use case were already detailed in D8.2, but a short overview is given in the following section.

2.2.1 The Test Request process

Each customer of BT Wholesale needs to be able to raise a request for BT to perform a test on its network. This is an established process using XML documents as its message format.

Each message is the service providers request to conduct a test on BTs network or BT’s response to such requests. The process will return either the result of this test or details of any errors that occurred, e.g. the test has been rejected. If the request is rejected, the customer is provided with the failure reason. Figure 2 shows a sequence diagram of the current solution.

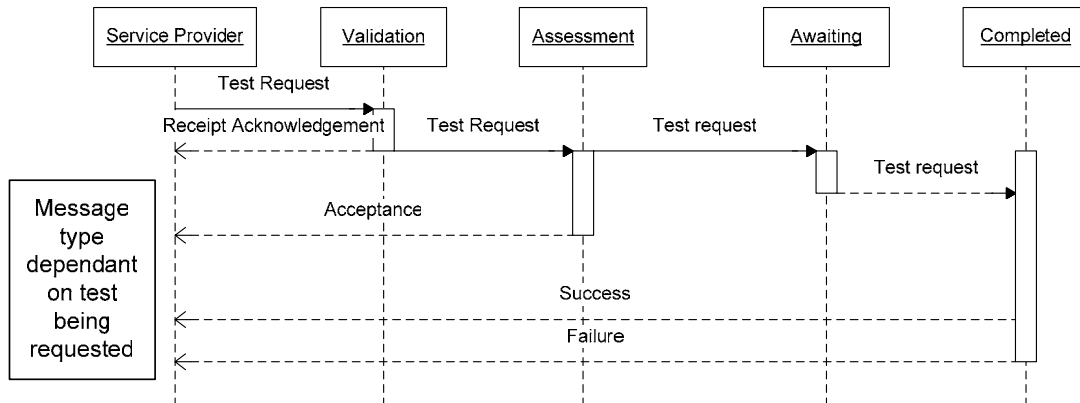


Figure 2: B2B Assurance Integration sequence diagram for any test request

2.2.2 Scenario description for the Test Request process

A high level use case scenario description for the first B2B Integration platform prototype is described below, along with several optional steps, which need to be performed if message conversion or mediation of message exchanges is necessary:

1. A Customer informs his Service Provider of an error occurring in one of his products through the Service Provider’s ticketing system. This action raises an error with the Service Provider.
2. The Service Provider then produces a message in a specific format. The message contains the data payload (e.g. the product to be tested, an error classification).
3. The Service Provider sends the message to the B2B Integration Platform.
4. The B2B Integration Platform performs the requested test by invoking the BT Wholesale’s responsible web service. This is hard coded in the current prototype, but in a wider scope would be discoverable.
5. After performing data mediation, the B2B Integration Platform invokes the selected service.
6. BT Wholesale’s, web service performs the requested operations on the Service Provider’s telecommunications equipment.

Extensions:

- 4a: The B2B Integration Platform has to perform a message conversion on the Service Provider’s request, as the message is not supplied in WSML.
 - .1: B2B Integration Platform delegates the message conversion to a responsible Adapter component.
 - .2: The Adapter sends the converted message back to the B2B Integration Platform.

4b: The B2B Integration Platform has to mediate between the message exchange pattern requested by the Service Provider and the pattern used by BT Wholesale's web service.

.1: B2B Integration Platform delegates the message exchange mediation to a responsible Choreography component.

.2: The Choreography component mediates between the different MEPs.

5a: The B2B Integration Platform has to perform a message conversion on the message to be sent to the selected service, as the web service cannot work with WSML messages.

.1: B2B Integration Platform delegates the message conversion to a responsible Adapter component.

.2: The Adapter sends the converted message back to the B2B Integration Platform.

2.3 Overview of the other functions performed by the eCo Platform

The eCo Platform provides functionality to carry out assurance related operations on BT's network. These are primarily split into four areas:

Network Testing

BT provides wholesale network access, which third party suppliers then use to provide other services to their customer (e.g. ADSL). If their customer reports that there is a problem with their service, the supplier needs to identify which part of the network that problem resides on. If they suspect that the problem is with the BT part of the network they will request a test to see if it is.

Functions

RequestTest

Fault Identification/Reporting

If a fault is identified with the network, the third party supplier will require that BT identifies the source of the problem and reports back to the supplier on the nature of the problem.

Functions

RequestTroubleReport

Fault Correction

If the problem is identified to be with BT owned network or equipment that cannot be resolved automatically, then it is necessary to arrange for an engineer to fix the problem.

Functions

RequestAppointmentAvailability

MakeAppointment

CancelAppointment

Provision

If the supplier is providing a product that requires BT to test or activate equipment before it can fulfil a customers order (e.g. ADSL) then it will require BT to provision this. Similarly it will require them to cease this when the customer leaves

Functions

- ProvideLineTest
- ActivateLine
- CeaseLine

2.4 Message Conversion

The core of the B2B Integration platform, WSMX in our case, can work only with messages described in WSML. Therefore several steps of message conversion are necessary, once for the message sent from the service requestor, and again before the invocation of the back-end Semantic Web Service. Custom adapters convert the content of a message between the different languages used in the use case.

2.4.1 Scenario description

1. The B2B Integration Platform detects that a conversion of a message specified in one of the possible languages to another one is necessary.
2. The B2B Integration Platform selects a suitable Adapter component, and requests a message conversion.
3. The Adapter component returns a message containing the conversion to the B2B Integration Platform.

2.4.2 Sequence diagram

A general Message Conversion sequence between two given languages is shown in Figure 3, while an example for an actual Message Conversion as needed by the prototype implementation is provided in section 5.

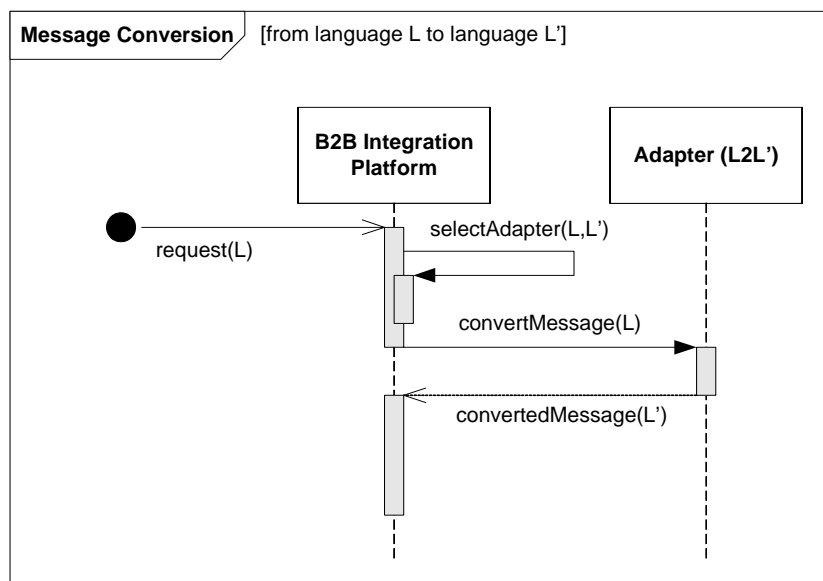


Figure 3 Convert Message sequence diagram

2.5 Ontology Mapping and Data Mediation

Given the domain and choreography ontologies of two parties that wish to exchange messages, the following section details the sequence of actions that lead to the production of a data mediator, by means of invoking a mediation service.

2.5.1 Use case diagram

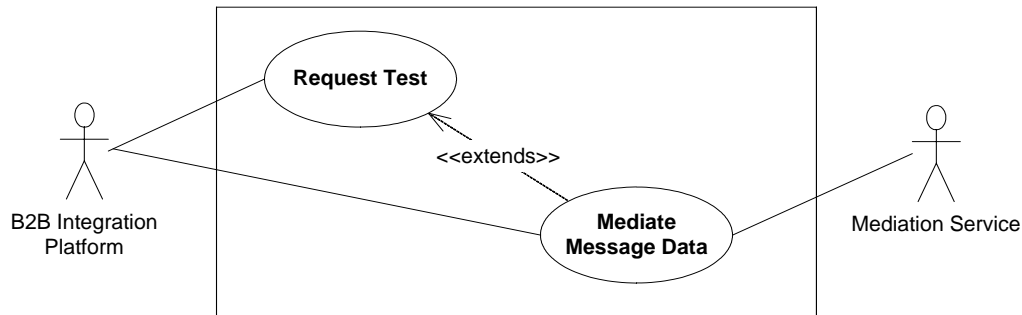


Figure 4: Data Mediation Use Case

2.5.2 Scenario description

1. The B2B Integration Platform gets a request necessitating the mapping of some ontologies, respectively of an instance set (I) based on a given ontology (O).
2. The B2B Integration Platform selects a suitable mediation service (which could be provided by a third party). The mediation service needs to be able to perform mediation between the source ontology (O) (e.g. the Service Provider's domain ontology) and the target ontology (O') (e.g. BT Wholesale's domain ontology).
3. The B2B Integration Platform Architecture then invokes the responsible mediation service, the submitted message containing both ontologies and the source data (I).
4. The Mediation Service sends the mediated data (I') back to the B2B Integration Platform.

2.5.3 Sequence diagram

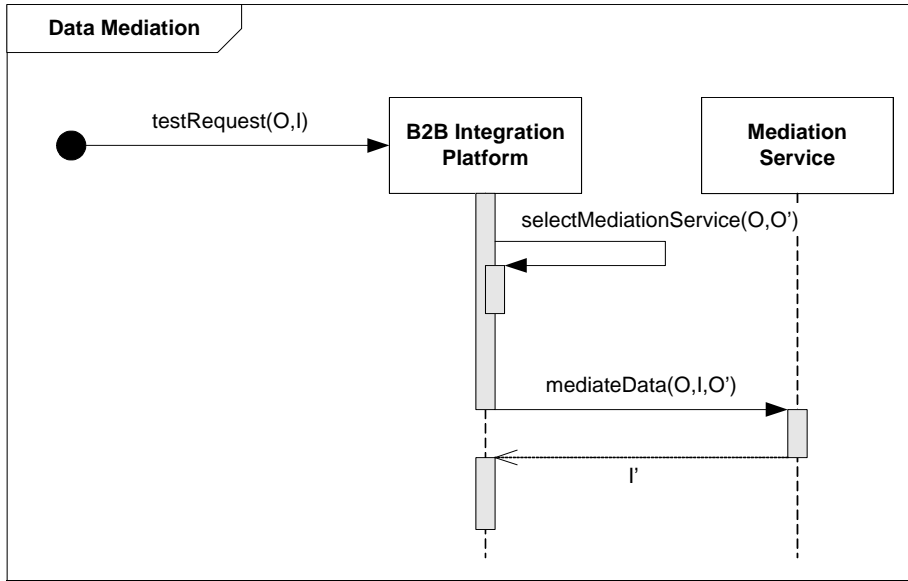


Figure 5: Data Mediation sequence diagram

3 Architecture of the Assurance Integration Prototype

This section presents the high level architecture for the Assurance Integration Prototype. This draws upon the components identified in the previous section. This follows the Design Guidelines specified in section 1.1.

In order to identify the necessary components for the prototype it is first necessary to consider the goal and intended use of the prototype. The goal is to show how semantically described interfaces can reduce the time and costs involved in integrating the heterogeneous OSS systems of partner companies. The intended use is to provide the means to mediate between the data and process of the interfaces at the semantic level and to apply those mediators to messages as they arrive (or leave) the platform. Thus the high level functionality required is:

1. Semantic representation of interfaces (both in terms of data and process) using WSMO
2. Creation of data mediators
3. Creation of process mediators
4. Adaptation of native messages to WSML format
5. Adaptation of WSML messages to native format
6. Data mediation of WSML messages
7. Process mediation of WSML messages
8. Invocation of service/interface

Items 1-3 are design-time activities. These could either be enabled by authoring directly in WSML or by using an appropriate GUI if available.

Items 4-8 are run-time activities which will make use of the output of activities 1-3 to allow valid interaction to occur. The sequence of events at runtime can be described as:

- A service interface sends a message in its native format e.g. XML in our example for Web Service 2 in Figure 7.
- An adaptor is applied to convert the message to WSML Adaptors will be used to allow existing web services from trading partners to integrate with the B2B Integration Platform.
- The description of the appropriate target interface is retrieved from the data store.
- Data and process mediation is applied to message exchanges to implement an interoperation layer among heterogeneous services.
- Outgoing messages are adapted to the native format of the destination interface in the case of the BT Wholesale eCo platform this will be in ebXML.
- The outgoing message is forwarded to the intended destination.

Figure 6 shows the high level architecture for the prototype including the necessary DIP Architecture components. This includes a set of design-time tools to create the semantic descriptions of the services and mediators, the WSMX core, communication manager and resources manager to manage storage and interaction, the adaptor framework to

convert to and from WSMML and mediator components to apply the WSMO mediators at run-time.

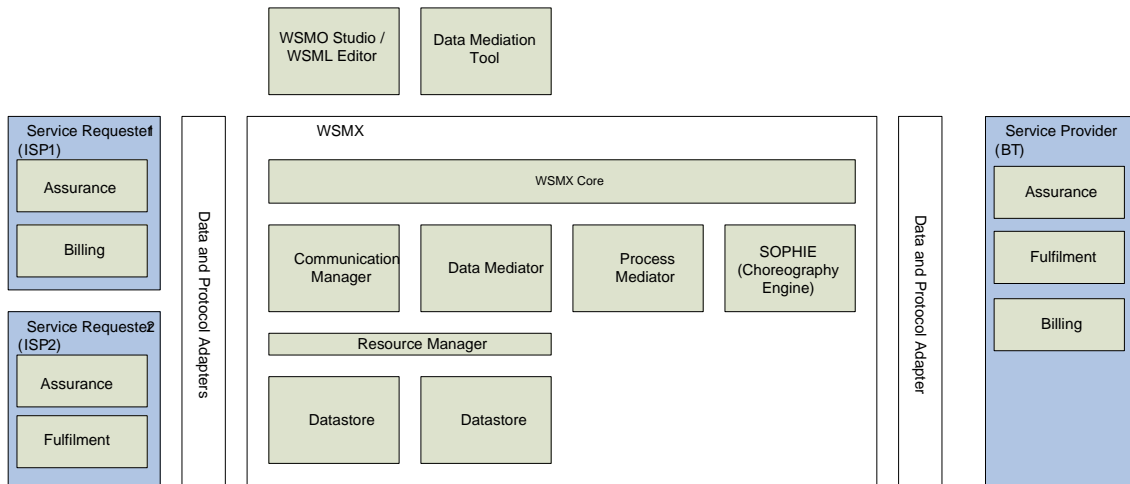


Figure 6: WSMX being used in the Telecommunications B2B Integration Platform

Grounding this in the use case, the Service Provider, BT, publishes its B2B integration services (including Assurance, Fulfilment and Billing services) whose data and processes are adapted from ebXML into WSMML for the DIP Architecture. Service Requesters such as Service Requester 1 (ISP1) then use messages generated by their web services, such as their Assurance and Billing Services to make service requests to BT [4]. To allow each of these businesses to operate with existing message formats and process flows, adapters lifting these schemas to WSMML will allow the tools provided by WSMX to be used, as discussed above. These WSMML representations of the messages form instances of ad hoc message ontologies that mirror the format of the messages (defined in this case by XML Schema). Mediators within the DIP Architecture will then perform any semantic mediation required between the differing ontologies. The ad hoc message ontologies can be related to a domain ontology in order to provide context for message elements. For example, a ‘custID’ field in an XSD would be mapped to an ontological property in the ad hoc WSMML message ontology. By itself, this property has no additional meaning over the XSD representation. Only by making it equivalent to the property ‘ID’ of the concept ‘customer’ in the domain ontology will it be given a wider context. This context can be used to determine appropriate mappings between differing message elements.

The adapters and mediators will be necessary for the defined relationship between the service provider and ISP to function accurately. These components must be produced during design time. For the initial prototype, mapping of source to target formats will also be performed during design time. At run-time defined parameters will be read and processed with the use of the DIP Architecture components. The Adapter Framework and use of Adapters within the Assurance Integration prototype is examined in greater detail in Section 4.1.3.

Choreography will allow process adaptation to occur, allowing differing process models to be used by the communication parties. Two semantically described choreographies can be evaluated in order to determine if they can communicate (either directly or via a process mediator) or not. If a mediator is required, it should be created and applied at run-time as appropriate messages arrive. In the initial prototype, the choreography will be semantically defined at design-time in order to allow process mediation to be carried

out on incoming and outgoing data messages at run-time. SOPHIE will be used to describe the defined choreographies and carry out process mediation.

While discovery will not be a feature in the initial prototype, service registration and discovery would allow much of this functionality to be published by the Service Providers and ISPs and facilitate the formation of ad-hoc relationships. These will allow the existing choreographies to be identified at run-time and process alignment to occur through use of a process mediator.

It is expected, if a SWS approach such as WSMX is adopted throughout the telecommunications industry, the requirement for adapters will be short lived where-as mediators will form an essential part of WSMO and business relationships. Adapters will be owned by the ISPs. Part of the relationships initially formed by ISPs and Service Providers may well include some consultancy in the production of adapters or even complete provision of them.

4 Assurance Integration Design

Building upon the design decisions from section 3, a more concrete design for a specific use case – the Assurance Integration use case – is detailed in this section. The first part of the section further details the use case, updating the involved actors and providing use case and sequence diagrams for all processes which are part of an usual Assurance Integration scenario. The second part of this section details the actual design decisions for the additional components necessary to integrate the DIP Architecture in a functional B2B Integration Platform prototype. In addition, sample WSMO service descriptions, message formats and message exchange patterns of this application scenario are provided in the prototype design.

4.1 Design of interface components

4.1.1 Web Services

For the first prototype simple web services will be created that mimic the function of the interfaces provided on the B2B Integration Platform. These will interface with a visual indicator which will provide information about the current state of interaction as a simulation of interaction with back-end systems.

4.1.2 UI and Front-ends

In order to provide a fully functional first prototype and to be able to demonstrate the integrated components, an understandable user interface is required. The Front-end designed for the Assurance Integration prototype simulates a possible real world Service Provider, i.e. the ticketing system provided by the Service Provider to allow for customer support. Customers who need to report an error, would use this part of the Service Provider's website to fill in several forms, describing the sort of error, as well as additional customer based information, such as the type of the product that is malfunctioning. Submitting the form would then raise an error with the Service Provider, starting the "Test Request" process.

The web application running on the Service Provider's machines takes the submitted form values as input for the construction of a suitable XML message, according to its own business domain ontology. The message includes the data payload, as provided by the customer. Once ready, the message would be supplied to BT Wholesale, respectively an instance of the B2B Integration platform, which takes over further processing as detailed in Section 3.

For the design of the web application, two parts can be distinguished:

- A presentation layer, providing a form based user interface for customers to request support for a certain kind of error. The user interface should correspond to the look and feel of a Service Provider's ticketing system, to simulate a possible real world scenario.
- An application layer, collecting user supplied information and responsible for constructing the XML message, which is then sent to the B2B Integration platform.

The design of the front-end should be based on a simple MVC architecture. For ease of conversions and necessary transformations, a Java/XML based architecture should be considered¹.

4.1.3 Adapters within the Assurance Integration Prototype

As the prototype in D8.4 is based on existing systems, it requires adapters to convert its messages into WSMML for the DIP Architecture and back out for the eCo platform and trading partners web services. The eCo platform requires its messages in ebXML, where-as a variety of message formats are being examined for the Service Provider web services.

Thus, Figure 7 shows the adapters required by the initial prototype: the differing message formats demonstrate the requirement for data adapters.

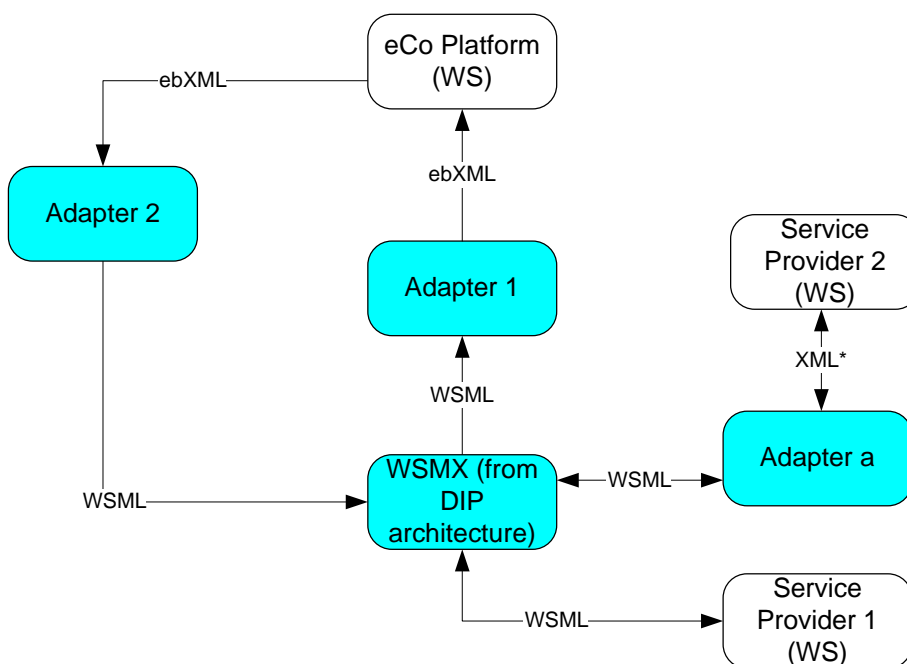


Figure 7: Message Flow and Adapters Required in D8.4

Since the WSMX/DIP Architecture environment requires messages to be presented in WSMML, adapters must be used for the eCo Platform web services and those of the Service Providers 1 & 2.

The eCo Platform uses ebXML for its messages and hence Adapters 1 and 2 must exist to translate these messages to and from WSMML. The resultant WSMML will then be mediated by the DIP Architecture to be used by “Service Provider 1” and “Service Provider 2” web services (the ebXML format was examined in DIP 8.2, [4]).

Adapters transform the received message from its source format to the target format. The transformation is concerned only with the syntactic mapping between message formats, leaving the semantics of the message intact. The semantic part is enforced by the ontologies that are used to build the outgoing message format.

¹ Such as Cocoon, an open source XML publishing framework. (<http://cocoon.apache.org/>)

The main problem tackled by the WSMX adaptor framework, at present, is the transformation from UBL (XML) to the format comprehensible by WSMX (WSML). The syntactic mapping between ontologies in different formats is done by extracting the instances from the source ontology fragment, then mapping them and finally populating the target WSML ontology fragment, [4].

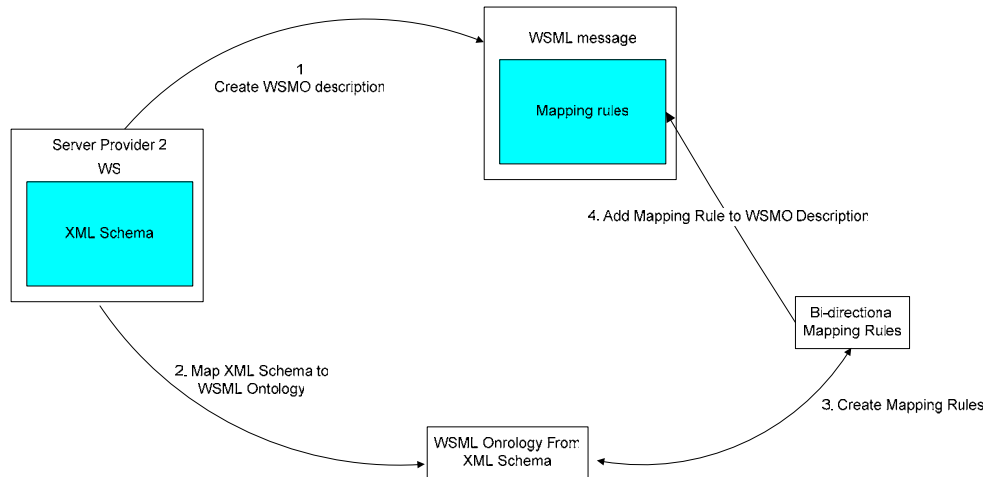


Figure 8: Overview of lifting and lowering XML messages into WSML messages

Figure 8 demonstrates the high level process of lifting, e.g. an XML message produced by an ISP into WSML [5].

Step 1 indicates the creation of a WSMO description for the service. Step 2 is the creation of an ad-hoc ontology that is equivalent to the XML Schema used by the web service. This step is intended to be automatic based on mappings between the conceptual model for WSMO ontologies and the conceptual model for XML Schema. Step 3 is the creation of mapping rules based on the mappings mentioned for step 2. Finally, the mapping rules should be stored as part of the instantiated message or stored as an adaptor for use at run-time to map XML messages to their WSML equivalents. Figure 9 shows the use of the Adapter Framework, [4] to apply such adaptors to XML messages.

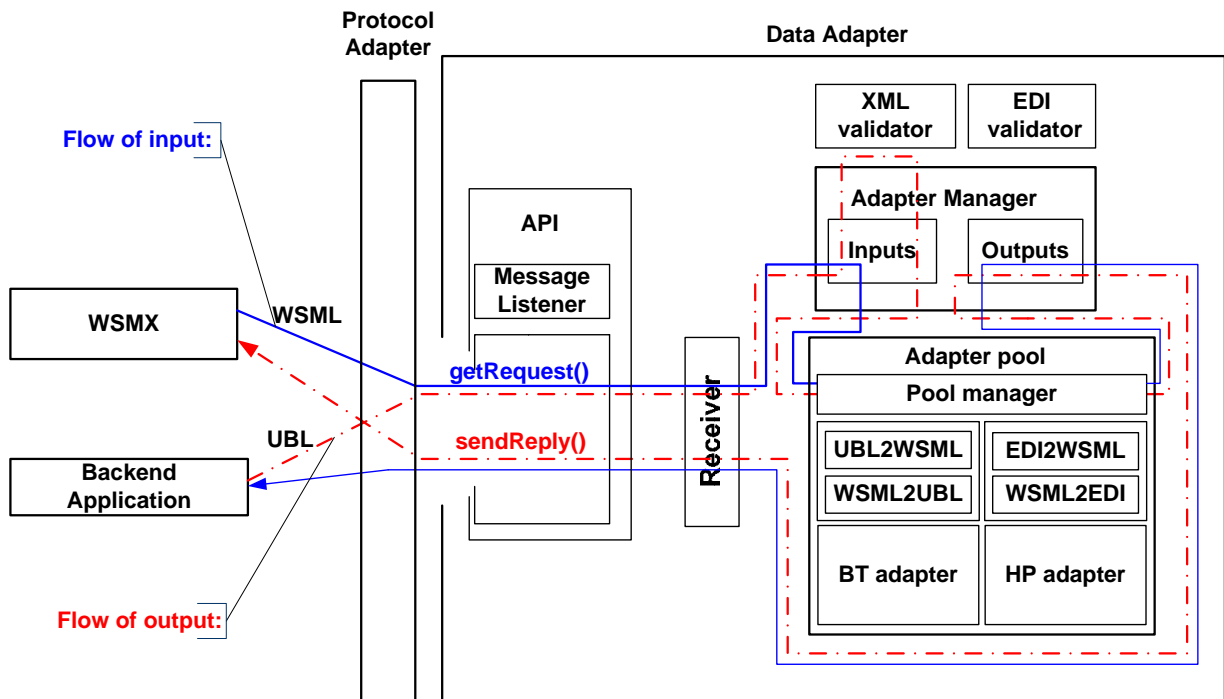


Figure 9: Adapter Framework

In Figure 9 the operation of the Adapter Framework, from [6], can be described as:

1. Taking what happens with the “flow of the output” message (indicated by the red / dotted line):
 - The “Backend Application” sends UBL document invoking the getRequest operation of the Adapter Framework API which is then received by the Receiver. The Receiver here is nothing other than an implementation of Adapter Framework API.
 - UBL document is then passed to Adapter Framework Manager which validates it. If it is indeed an XML document then the Manager forwards it to the Adapter Pool Manager. If the validation fails, MESSAGE.FORMAT_ERROR is returned back to backend application.
 - Adapter Pool Manager selects an appropriate adapter (i.e., UBL2WSML), that converts the UBL document to a WSML document. This WSML document is then forwarded to Adapter Framework Manager.
 - Adapter Framework Manager then sends this document to WSMX through the sendReply operation of Adapter Framework API.
2. Taking what happens with the “flow of the input” message (indicated by the blue / solid line):
 - WSMX sends a WSML document invoking the getRequest operation of Adapter Framework API which is then received by the Receiver.
 - This WSML document is then passed to Adapter Framework Manager which simply forwards it to Adapter Pool Manager. Here, no validation takes place,

at this moment, as the Adapter Framework trusts WSMX and always expects WSMML documents from it.

- The Adapter Pool Manager then selects an appropriate reverse adapter (i.e., WSMML2UBL), that converts the WSMML document to a UBL document. This UBL document is then forwarded to the Adapter Framework Manager.
- The Adapter Framework Manager then sends this document to the backend application through sendReply operation of Adapter Framework API.

While the prototype will be able to use the framework to form the adapters, the proposed means of producing them is as one-off components produced by WP8 with consultancy from WP6. Similarly, it is expected that while individual companies will generally build their own WSMML adapters they may buy these from providers or buy consultancy on how to produce them.

For D8.4, it is expected that the prototypes will use standard Java API's to parse and transform the XML messages into WSMML. Ideally, this would be available through the DIP Architecture although there are no plans to provide this beyond the framework in WP6 for D8.4.

4.2 Services

4.2.1 Domain Ontology

A domain Ontology is used to represent localised or user specific information. In this case the domain ontology is a conceptualisation of the data passed between the users of the assurance integration functionality of eCo. It provides an important bridge between ebXML messages used by the actual web services in the eCo Platform and higher level ontologies which represent a more generalised view of data.

The ontology defines concepts in a way that allows the original ebXML messages to be reconstructed, but also enables them to be linked to concepts in higher level, or customer specific ontologies. The process of lifting and lowering messages from their original XML form into an ontological representation, via adaptors is one of the important aims of the first prototype.

For example below is a fragment of the ebXML message that represents an error message

```
<dt:ListOfTestResponseDetail>
  <dt:TestResponseDetail
    <TestCategory="IncidentCheck">
      <dt:TestFeatures>
        <dt:TestOutputFeatureSet>
          <dt:ErrorInfo>
            <dt:Code>00101</dt:Code>
            <dt:Msg>Service ID is an invalid value</dt:Msg>
          </dt:ErrorInfo>
        </dt:TestOutputFeatureSet>
      </dt:TestFeatures>
    </dt:TestResponseDetail>
  </dt:ListOfTestResponseDetail>
```

This conveys two classes of information, the category of the test and the details of the error message. These are therefore modelled as two concepts, *TestCategory* and *ErrorMessage* (shown below).

Concept TestCategory

```

    Category ofType XSD:String
    nonFunctionalProperties
      dc:description hasValue "Type of Test to be carried out, e.g OneShot"
    endNonFunctionalProperties
Concept ErrorMessage
  Code ofType XSD:Integer
  Message ofType XSD:String
  nonFunctionalProperties
    dc:description hasValue "Error Message and Code"
  endNonFunctionalProperties

```

Relationships defined between concepts allow us to model how they are connected in terms of the original ebXML messages but also how they relate to other more general concepts in the domain ontology and other higher level ontologies. For example the Concept `TestRequest` defines the relationship `hasTestCategory` between itself and `TestCategory`. This tells us that `TestCategory` is part of the `TestRequest` message sent by the service provider. These relationships may be seen in Figure 10 below.

As these domain specific concepts relate very closely to the format of the ebXML messages sent in the system it is also important to define relationships between more general higher level ontologies. In the Telecommunications Sector the TMF has defined a data model known as the SID which provides a general model of data and business entities used in telecommunications.

Where possible, concepts in the domain ontology that can be related to entities in the SID have been defined such that they either directly subclass a SID concept and then extend where necessary, or define relationships between existing SID concepts. For example the SID defines two concepts `Party` and `PartyRole`. The concept `Party` is used to explicitly define an organisation or individual and `PartyRole` allows an organisation/individual to take on a particular role during a business transaction. On the Eco platform these concepts fit nicely, as there are a number of organisations that use the platform (such as BT and other third party providers) but they also take on different roles depending on the operation being undertaken. If a third party provider wishes to carry out a `testRequest` operation, then the Concept `Party` is used to describe their organisation, and `PartyRole` is used to define their role in this transaction as “Conductor”. Similarly BTs `PartyRole` in this operation is “Performer” as they are performing the actual test.

By defining relationships between the SID it allows the domain ontology to provide information that is useful in mediation between other ontologies that use the SID. Also by defining concepts that closely reflect the structure of the messages on the eCo platform it allows messages to be easily lifted and lowered via adaptors.

The diagram in Figure 10 gives an overview of the eCo ontology in graphical form. Although the Ontology is expressed in WSMML, for the diagram it has been recreated with Protégé OWL plug-in to enable easy visualisation. The Concepts are shown in the beige rectangles with the subsumption relationships shown by a triangular arrowhead pointing to the super concept. Other relationships are shown with the property on the vertex.

The full domain ontology can be found in Appendix 1.

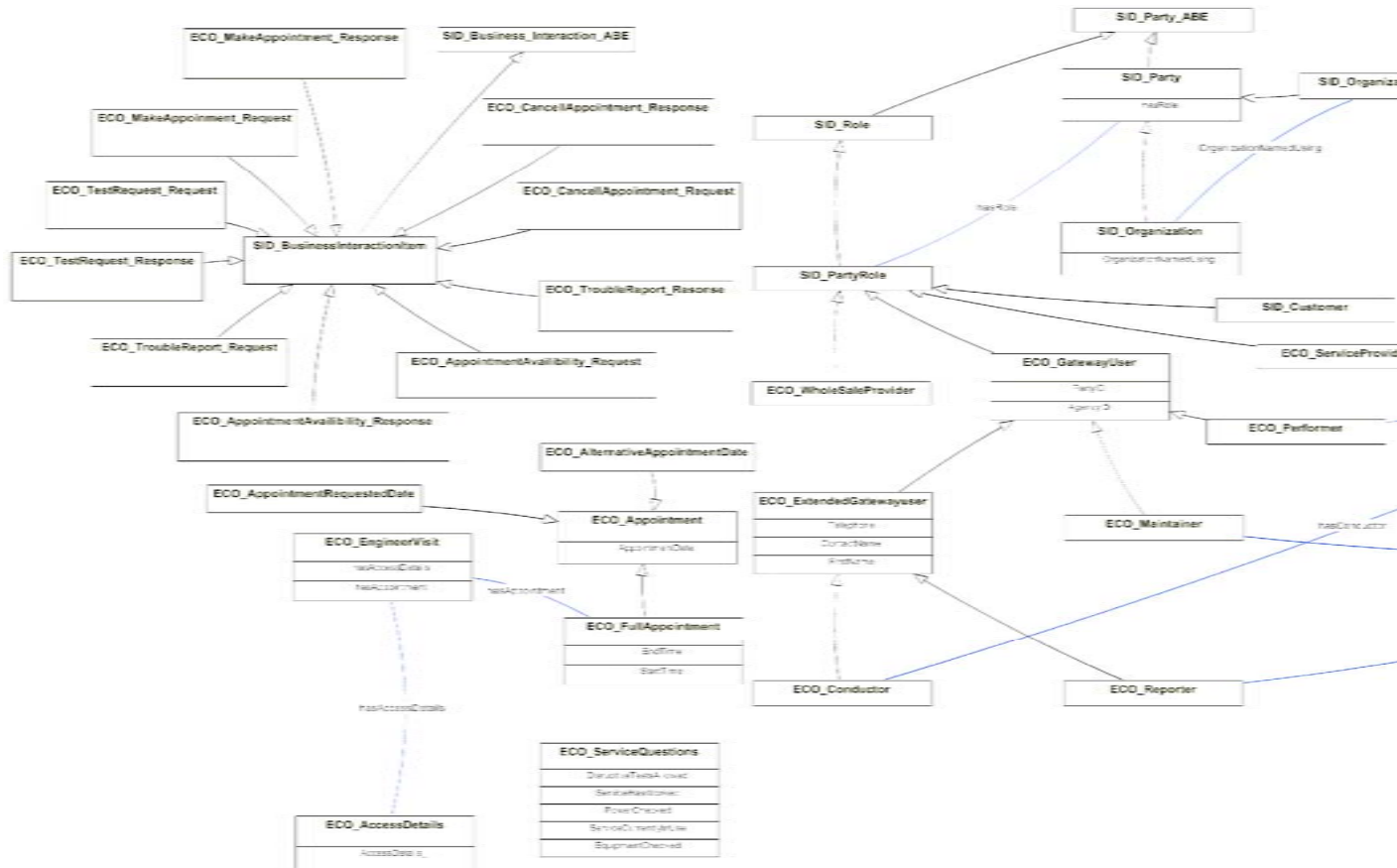


Figure 11 - Enlarged Eco Ontology (part 1)

4.2.2 WSMO Descriptions

4.2.2.1 Goals

In this use case the first prototypes primary aim is to show that representing data and services at an ontological level has significant advantages over previous non-semantic XML implementations, where the semantics of a message are unclear and hence the integration task is far harder and time consuming.

Goals in the context of WSMO are primarily a way of expressing a particular desire of a service requester, expressed in terms of ontologies and logical expressions. Goals also contain the interface of the requester (i.e. its choreography) which among other things is the means by which an appropriate process mediator is chosen, if required. This enables services to be discovered by matching their capabilities to the goal. With the eCo platform, discovering particular services is not the first priority in the case of Assurance Integration because a new eCo user will already know what services are available. The main problem is making sense of the data/messages that the service provides and integrating this with their own systems, which may have different interpretations of the data/messages.

It is therefore debatable whether goals are needed in this scenario, as the main purpose of them is not particularly relevant. It could be argued that a goal could be used to express the desire of a service provider, in terms of what functionality they require from an eCo service to be able to interface their internal systems. This may work if the service provider's goal exactly matches the capability of an eCo service, but in practice this is highly unlikely. Depending on the sophistication of the discovery and mediation components, it may be possible to have partial discovery which will match and link some elements of the Service provider goals with the eCo platform's service capability.

Other than discovery, goals may allow greater de-coupling between the client and provider. There may be situations in which the services or messages on the eCo platform change, meaning that the corresponding WSMO definitions would also have to be updated. If the trading partner used goals then it would enable them to remain somewhat oblivious to this, as it would only require updating the mediation between their goal request and the new eCo service. Also, goals allow for the system to scale more effectively. In the real system there are likely to be many diverse trading partners. Having goals allows us to model the problem in a consistent manner and would enable reuse amongst goal and mediator definitions. A new trading partner joining the eCo platform would be able to use one of the goal/mediator descriptions available and extend/adapt this to meet their requirements. This however is reliant on the Trading partner being 'WSMO aware', i.e. they have a domain ontology to represent information in their system and are prepared to model their requests in terms of WSMO. For the first prototype, it is assumed that the current state of play is that most trading partners do not use ontologies and instead have a proprietary messaging format based on XML. This means that goals will not be used in the first prototype, but as it is developed further the possibility of using goals will be examined. In addition, when a WSMX discovery component is available, it will enable full decoupling of the requester and provider by providing (albeit trivial) discovery to occur, and hence coupling, at run-time.

4.2.2.2 Web Services

The eCo Platform provides functionality to carry out assurance related operation on BT's network. These are primarily split into three areas:

- Network Testing
- Fault Identification/Reporting
- Fault Correction

For each distinct operation a WSMML description has been produced describing the capability that the service provides. The data elements are described in terms of the domain Ontology, named the “eCo” ontology. WSMML allows Web Service descriptions to be complex in nature, providing axioms to describe the pre and post conditions of the service. Some axioms have been included, placing conditions on allowable input to the services, but in general complex axioms have not been included. This is due to the current lack of support for reasoning, and higher emphasis on mediation and choreography in this use case.

Example

This section gives a detailed description, with underlying design rationale, of how WSMML descriptions are produced. The section examines the process with the example service *TestRequest*, which is used by a service provider to initiate a request for a test to be carried out in BT's network.

WSMML service descriptions are separated into two parts – a functional part, describing what the service can actually achieve (service capability) and a non-functional part, providing additional information about the service (non-functional and quality of service properties).

Non-functional Properties

The non-functional properties for *TestRequest* use the Dublin Core Metadata elements. These can be extended to include, for example, quality of service information. The property of most interest here is `dc:subject`, which has the value of an element within the TeleManagement Forum's eTOM² model: `eTOM:EvaluateAndQualifyProblem`. This allows a direct link to be made between the service and a specific eTOM category. The eTOM prefix is an XML namespace which is shorthand for the location where the eTOM ontology is defined. ‘Evaluate & Qualify Problem’ is an eTOM category in the ‘Service Problem Management’ process grouping. If appropriate, several eTOM categories can be attributed to a service using the subject property.

nonFunctionalProperties

```
dc:title hasValue "Request Incident Check Web Service"
dc:creator hasValue "Marc Richardson"
dc:subject hasValue "eTOM:EvaluateAndQualifyProblem"
dc:description hasValue "Will Initiate a request for an
incident check on the BT eCo platform"
dc:publisher hasValue "British Telecommunications PLC"
```

² <http://www.tmforum.org/>

```

dc:contributor hasValues {"Marc Richardson"}
dc:date hasValue "2004-12-15"
dc:typehasValue<<http://www.wsmo.org/2004/d2/#webservice>>
dc:format hasValue "text/html"
dc:language hasValue "en-us"
dc:rights hasValue <<http://www.bt.com/privacy.html>>
version hasValue "$Revision: 1.0 $"

```

endNonFunctionalProperties

Functional Properties

The *TestRequest* service could be described as having the capability that it initiates requests for diagnostic tests to be carried out over BT’s network. In WSMO, modelling preconditions and postconditions provide the conditions over the information space that must hold before and after the execution of the service. Capability assumptions and effects also define conditions but describe the state of the world instead (i.e. not directly associated with the information space but more likely with the physical world). As part of the modelling of WSMO services it is therefore necessary to identify the expectations and the influences of this service on the information space and the state of the world.

The inputs to *TestRequest* are detailed in **Table 1**.

Input	Description
Performer	The identifier for the performer who is carrying out the test. Each one is identified by two IDs, agencyID and partyID
Conductor	The identifier for the Conductor who is requesting the diagnostic test. Each one is identified by two IDs, agencyID and partyID. Contact details such as telephone number are also required
TestReference	The reference that identifies a particular test for the Service Provider, as well as the date of test
TestCategory	The type of test requested

Table 1 Inputs to *TestRequest* transaction

Preconditions of this service would be that the inputs supplied are all valid. The validity of these can be determined by evaluating defined axioms. The concepts used in these axioms can be related to ontologies in a similar way to the non-functional subject property. This allows their meaning to be explicitly stated rather than relying on a local interpretation. For example, the *partyID* input which refers to the service provider can be linked to the SID which contains a ‘party’ entity with a ‘partyID’ attribute and can be related to a role entity by the ‘hasPartyRole’ association which can be one of several types including ‘Service Provider’. By linking the service descriptions in this way, both humans and computers can understand more about what the *partyID* is in this case. Furthermore, because of the explicit link, the condition can be evaluated against existing

knowledge and data that have been described in the same fashion e.g. to check whether the *partyID* is indeed a valid ID of a service provider. The evaluation of a precondition for *partyID* is described below.

The following statement first checks that the *?partyID* input is an integer and then that the *partyID* of the Performer is not the same as the *partyID* of the Conductor (as they cannot both be the same)

```
?inputPerformer memberOf eCo:Performer [
    eCo:PartyID hasValue ?inputPerformerPartyID
] and
?inputConductor memberOf eCo:Condcutor [
    eCo:PartyID hasValue ?inputConductorPartyID
] and
?inputConductorPartyID memberOf xsd:integer and
?inputPerformerPartyID memberOf xsd:integer and
?inputConductorPartyID <> ?inputPerformerPartyID
```

Postconditions of the service can be described in a similar way and would in this case be used to determine that the output of the *TestRequest* was valid.

To enable definition of assumptions and effects of the *TestRequest* service the state of the world should be modelled. In this scenario a *TestRequest* could have one of four states:

Ready – The test request is ready for submission.

Awaiting – The test request has been accepted but the test has not been completed yet.

Completed – The test has been completed

Failed – The test request was not accepted and no test was carried out.

The states are modelled through variables, and the values of these indicate the state of the world at a given time. Again, the world states should be expressed in an ontology to which the axioms then refer. The state of the world in which *TestRequest* can operate (assumption) is one where the output *TRstate* has the value “*Ready*”. The execution of *testRequest* changes the world state (effect) so that *TRState* value changes to “*Completed*”. The WSMO axiom defining this assumption can be defined with the following expression:

```
?WState memberOf WorldState and
?WState [TRState hasValue “Ready”]
```

where *WorldState* is a predefined concept having *TRState* as an attribute. The effect is similarly defined:

```
?WState memberOf WorldState and
?WState [TRState hasValue “Completed”].
```

Thus WSMO can be used to describe a rich set of properties that the service has including its capability, its data requirements for input and output and its effect of the information space and state of the world.

The full WSMML descriptions for this service along with others can be found in Appendix 1.

4.2.2.3 Mediators

Mediators will be defined using the ontology mapping language developed in the DIP and SEKT projects. This has been implemented by a WSMX mediation component which includes a design-time UI to create a mediator between two WSMO ontologies and a run-time component to execute it.

The design time component enables you to load two ontologies and to create mappings between these two ontologies. The mappings will be stored as mappings in the abstract mapping language developed in DIP/SEKT .

The Run-time component takes the previously created mappings, applies the grounding and executes these mapping rules on the incoming instances (source instances) and produces the target instances.

For the prototype in 8.4, mediation is required between BTs domain ontology (eCo) and the third party trading partner ontologies. This will require a mapping to be created between concepts in both ontologies that represent the elements in the underlying messages being sent and received by each party. An example is given below to demonstrate the steps required for mediation.

The trading partner ontology (tPOnt) contains a concept *Customer* with a property called *customerID*. The BT ontology (eCo, see Figure 10) contains an equivalent concept *Party* with a property called *PartyID*. The design time mediation tool will be used to define a mapping where *tPOnt:Customer* is expressed as an equivalent concept to *eCo:Party* and were *tPOnt:customerID* is an equivalent property to *eCo:PartyID*.

At runtime the steps required for a trading partner to send a message containing the field *custID* are as show below. Steps 3-5 show the data mediation taking place:

1. The trading partner sends an XML message containing a field *custID*.
2. An adaptor is applied to the incoming message which creates a WSMML message where the *customerID* property has the value of *custID*.
3. The WSMML message is then passed to the runtime mediator which holds the mapping rules that were defined between the tpOnt ontology and eCo ontology
4. The runtime mediator see that CustID is and Instance of cutomerID in tpOnt ontology so looks for a mapping rule for this. It finds that CustomerID in tpOnt ontology maps to PartyID in the eCo ontology.
5. The runtime mediator then applies the mapping rules to the tpOnt WSMML message to create an eCo WSMML message, such that PartyID has the value of CustID
6. The eCo WSMML message is then adapted to the XML format of the BT interface.

The mapping rules defined at design time and executed at runtime are actually part of an ontology, as are all the basic entities of WSMO. The question arises of where this ontology should be placed. We could include it as part of the BT domain ontology, as part of the trading partner ontology or we could put it in a separate mediation ontology. For the first prototype we will only be defining mediation between BT and one trading partner, so the location of the mediator definition is not greatly important. At some point however when the system is expanded to incorporate many trading partners it may make sense to pool all of the mediator definitions together, so that we can have some reuse between them (via subclassing). It is likely therefore in the future as the system becomes more mature that a separate dedicated ontology for mediation will be created.

4.2.2.4 Choreography - SOPHIE

In defining the choreographies of the Service and Wholesale Providers, the paradigm presented by SOPHIE [3] has been followed. SOPHIE defines a conceptual framework, which in a first cut is divided into syntactical and semantic models. The syntactical model details the syntax of the framework as three different complementary models, namely: structural, behavioural and operational. The structural model deals with the provision of a reusable collection of entities following different levels of abstraction that facilitate the basis for the description of a conceptual model. The behavioural model cares for the description of the dynamic interaction among the entities defined in the structural model. The operational model facilitates the means to allow the interoperation among different behavioural models.

Structural model

Table 2 details the building blocks of the structural model of SOPHIE.

<i>element</i> = [name, type, value]
<i>document</i> = [name, URI, elements* ³ , businessRules*]
<i>message</i> = [name, URI, from ⁴ , to [?] , documents*]
<i>messageExchangePattern</i> = [name, URI, description [?]]
<i>message exchange</i> = [name, URI, mep [?] , messages*]
<i>conversation</i> = [name, URI, messageExchanges*]

Table 2 Structural model of SOPHIE

Behavioural model

Table 3 details the building blocks of the behavioural model of SOPHIE.

<i>booleanExpression</i> = [name, URI, expression [?]]
<i>state</i> = [name, URI, subStates*]
<i>action</i> = [name, URI, task [?]]
<i>task</i> = [party.message]

³ The symbol "*" represents that there can exist zero or more instances of the attribute

⁴ The symbol "?" represents zero or one instances of the attribute

<i>event</i> = [<i>name</i> , <i>URI</i> , <i>booleanExpression</i> [?]]
<i>guardCondition</i> = [<i>name</i> , <i>URI</i> , <i>rule</i> [?]]
<i>rule</i> = [if <i>booleanExpression</i> then <i>state</i>]
<i>guarded transition</i> = [<i>name</i> , <i>URI</i> , <i>events</i> *, <i>guardCondition</i> [?] , <i>actions</i> *]
<i>part</i> = [<i>name</i> , <i>URI</i> , <i>messageExchange</i> [?] , <i>guardedTransitions</i> *]
<i>choreography</i> = [<i>name</i> , <i>URI</i> , <i>conversation</i> [?] , <i>parts</i> *]

Table 3 Behavioural model of SOPHIE

Finally, the semantic model provides the means to describe the structural and behavioural models, in order to produce the operational model as a result of a reasoning task. Appendix 2, presents the choreography ontology from which parties can inherit to define their own models.

4.2.2.5 Choreography Example

In order to define a choreography according to the paradigm presented in SOPHIE the next steps need to be followed.

Definition of the Structural Model

For each service capability, identify:

- Elements
- Documents
- Messages
- Message Exchanges and the MEP they follow
- Group message exchanges into conversations

Definition of the Behavioural Model

For each one of the identified message exchanges, define the behavioural model as an Abstract State Machine (ASM) [2]. Then it is required to define:

- Boolean expressions
- States
- Actions
- Events
- Rules
- Guard conditions
- Guarded Transitions
- Parts
- Choreographies

Definition of the choreography ontology

Using the previously defined domain ontology and the choreography model defined in appendix 2, it is possible to specify a choreography ontology that depicts the structural and behavioural models. To do so, we will import from both (domain and model choreography) ontologies, in order to define our models.

Example

The example applies the operational model of **SOPHIE** to the broadband test interface. Figure 13 shows a simplified message exchange pattern for the test interface. The service provider sends a *testRequest* message to BT. The requested test is examined and then accepted or rejected (it may be rejected if, for example, a test is requested on a line not owned by BT) resulting in a *testAccept* or *testReject* message being sent to the service provider. Following an acceptance notification, the test is carried out and the *testCompleted* message is sent which contains the result of the test. Should an error occur the message *signalException* is sent.

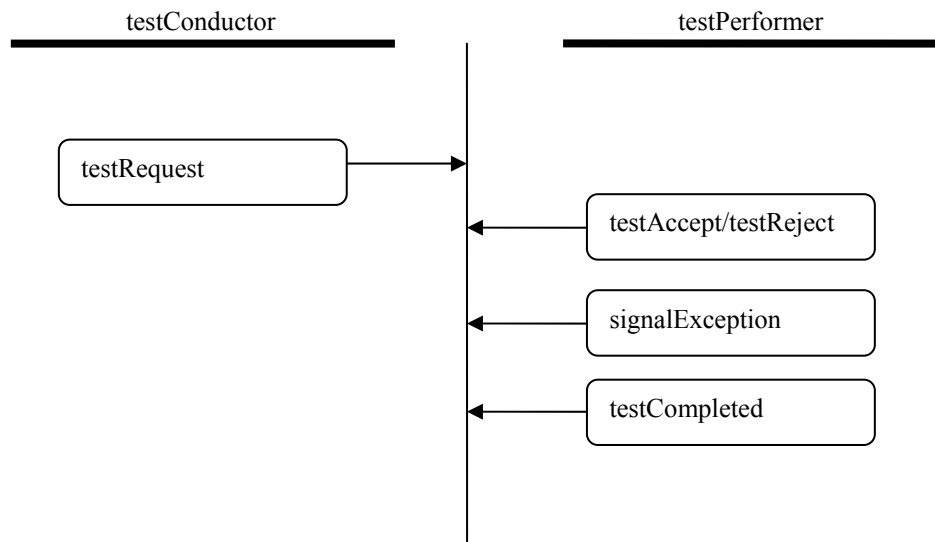


Figure 13: In-Multi-Out Message Exchange Pattern

If the service provider were to use a different MEP where it expects BT to provide a single response message (*Mcompleted*) indicating whether the test was accepted or rejected and if accepted the result of the test, then the operational model of SOPHIE can be applied. Figure 14 shows the MEP⁵s followed by interacting parties.

⁵ Rather than describing the internals of the process the ASM needs to describe the behavior of the MEP

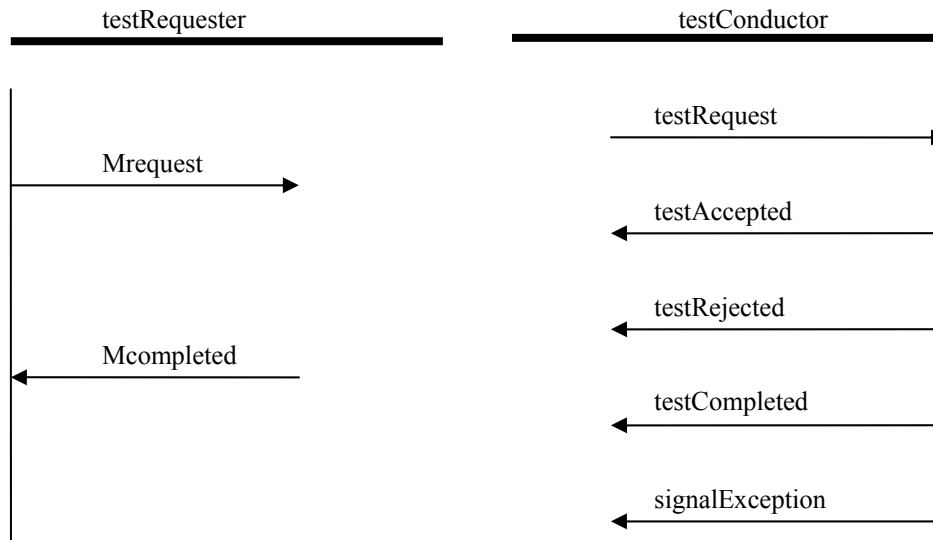


Figure 14: Request-Response and In-Multi-Out Message Exchange Pattern

The test requester uses follows the message exchange “*request-response*” which uses the following structural model:

- Elements: testDate, testType, partyID, testContact, contactName, telephone
- Documents: Drequest, Daccepted, Drejected, Dexception, Dcompleted
- Messages: Mrequest, Mcompleted
- Message Exchanges and the MEP they follow
- Mapping among documents and elements:
 - Drequest. {name, param1, param2, param3}
 - Dcompleted. { name, accept, except, result1, result2, result3}
- Mapping among messages and documents
 - Mrequest. {Drequest}
 - Mcompleted. {Dcompleted}

The test conductor uses follows the message exchange “*in-multi-out*” which uses the following structural model:

- Elements: testName, testParam1, testParam2, testParam3, testResult1, testResult2, testResult3, testAccepted, exception
- Documents: testRequest, testAccepted, testRejected, signalException, testCompleted, testAccepted
- Messages: testRequest, testAccepted, testRejected, signalException, testCompleted
- Message Exchanges and the MEP they follow
- Mapping among elements and documents:

- testRequest. {testName, testParam1, testParam2, TestParam3}
- testAccepted. {testAccepted}
- testRejected. {testAccepted}
- signalException. {exception}
- testCompleted. { testName, testResult1, testResult2, testResult3}
- Mapping among documents and messages
 - testRequest. {testRequest}
 - testAccepted. {testAccepted}
 - testRejected. {testRejected}
 - signalException. {exception}
 - testCompleted. {testCompleted}

For example, Table 4 shows a fraction of the domain ontologies λ and λ' used by the test requester and test conductor respectively, which detail the concepts “test” and “parameter”, and “testName”, and “testParameter” respectively.

<i>concept test</i>	<i>concept testName</i>
<i>nonFunctionalProperties</i>	<i>nonFunctionalProperties</i>
<i>dc#description hasValue "name"</i>	<i>dc#description hasValue "name"</i>
<i>endNonFunctionalProperties</i>	<i>endNonFunctionalProperties</i>
<i>concept parameter</i>	<i>concept testParameter</i>
<i>nonFunctionalProperties</i>	<i>nonFunctionalProperties</i>
<i>dc#description hasValue "in-out parameter"</i>	<i>dc#description hasValue "parameter"</i>
<i>endNonFunctionalProperties</i>	<i>endNonFunctionalProperties</i>

Table 4 Example of the domain ontologies λ and λ'

For example, Table 5 shows a fraction of choreography ontologies θ and θ' for the initiating party and answering service respectively, that depicts some elements, documents, boolean expressions and actions, used by the parties to model a message exchange.

<i>concept name</i>	<i>concept: testName</i>
<i>subconceptOf { Λ#element, λ#test }</i>	<i>subconceptOf { Λ#element, λ'#testName }</i>
<i>concept param</i>	<i>concept testParam</i>
<i>subconceptOf { Λ#element, λ#parameter }</i>	<i>subconceptOf { Λ#element, λ'#testParameter }</i>

Table 5 Example of the choreography ontologies θ , θ'

For example, taking as input the choreography ontologies presented in Table 5, the following relations among elements, documents, messages, boolean expression and actions are selected among the ones found compatible:

- $\{ name, param1, param2, param3, result1, result2, result3, accepted, except \} \approx \{ testName, testParam1, testParam2, testParam3, testResult1, testResult2, testResult3, testAccepted, exception \}$ that maps *name* with *testName*, *param1* with *testParam1* and so on.
- $\{ Drequest, DCompleted \} \approx \{ testRequest, testAccepted, testRejected, signalException, testCompleted, testAccepted \}$ that maps “*Drequest*” with “*testRequest*” and “*DCompleted*” with $\{ testAccepted, testRejected, signalException, testCompleted, testAccepted \}$ and vice versa
- $\{ Mrequest, Mcompleted \} \approx \{ testRequest, testAccepted, testRejected, signalException, testCompleted \}$ that maps “*Mrequest*” with “*testRequest*” and “*Mcompleted*” with $\{ testAccepted, testRejected, signalException, testCompleted, testAccepted \}$ and vice versa.

Taking as input this ontological mapping the operational model that allows overcoming the heterogeneities of the parties can be produced. In this case the test conductor’s response messages to the *Mrequest* message are merged and send to the service provider. A merge box as shown in Figure 15 permits to map the types and names of elements, documents and messages.

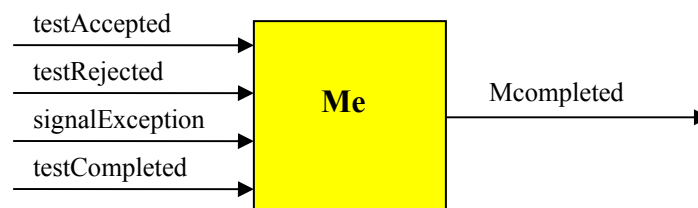


Figure 15: Merge Box for testReply message

From the point of view of the case study, SOPHIE provides a comprehensive way to semantically represent the choreography of the B2B Integration Platform's interfaces. With appropriate tool support, it can ease the integration problem and allow dynamic integration between partners to be realized. Tool support is required to automate the expression of ebXML Business Process Specifications into the ontological format of SOPHIE and to support to user in performing mappings between the choreographies of the different parties.

5 Implementation of the prototype

5.1 Assurance Integration Test Request

According to the description of the Test Request use case in Section 2.2, a more detailed scenario description for the prototype is given below. The extensional activities of Section 2.2 have been included in the main scenario description, as the prototype implementation needs both message conversion and the mediation of message exchange patterns. Following the scenario description, a sequence diagram for the main activities and actors is shown.

Scenario description

1. A Customer informs his Service Provider of an error occurring in one of his products through the ticketing system's user interface.
2. The ticketing system then produces a message in a specific XML format (including the data payload, describing the error and the Customer's product). The concepts referred to in this message conform to the Service Provider's domain ontology.
3. The message is sent to the B2B Integration Platform Front-end, which delegates the conversion of the message to the DIP Architecture.⁶
4. The DIP Architecture uses a custom XML2WSML adapter to perform the message conversion to WSML (the adapter should be designed following the adapter framework design guidelines supplied by WP6)
5. The converted message is consumed by the DIP Architecture.
6. The DIP Architecture invokes a Semantic Web Service able to fulfil the goal of testing a Customer's product and mediates by means of SOPHIE between the supplied Message Exchange Pattern and the MEP defined in the Semantic Web Service's WSMO description.
7. In addition the DIP Architecture performs data mediation if necessary.
8. The message exchange is performed, resulting in one or more invocations of the SWS according to the mediated MEP. This involves a second message conversion from WSML to ebXML, as the web service can only work with ebXML messages.
9. The result message is sent back to the Service Provider, after additional inverse data mediation and message conversion steps.

⁶ In practice the B2B Platform Front-end and the DIP Architecture are the same system. The Front-end exists as the interface that is seen by the Service Provider but utilises DIP Architecture components to achieve its aims.

Sequence Diagram

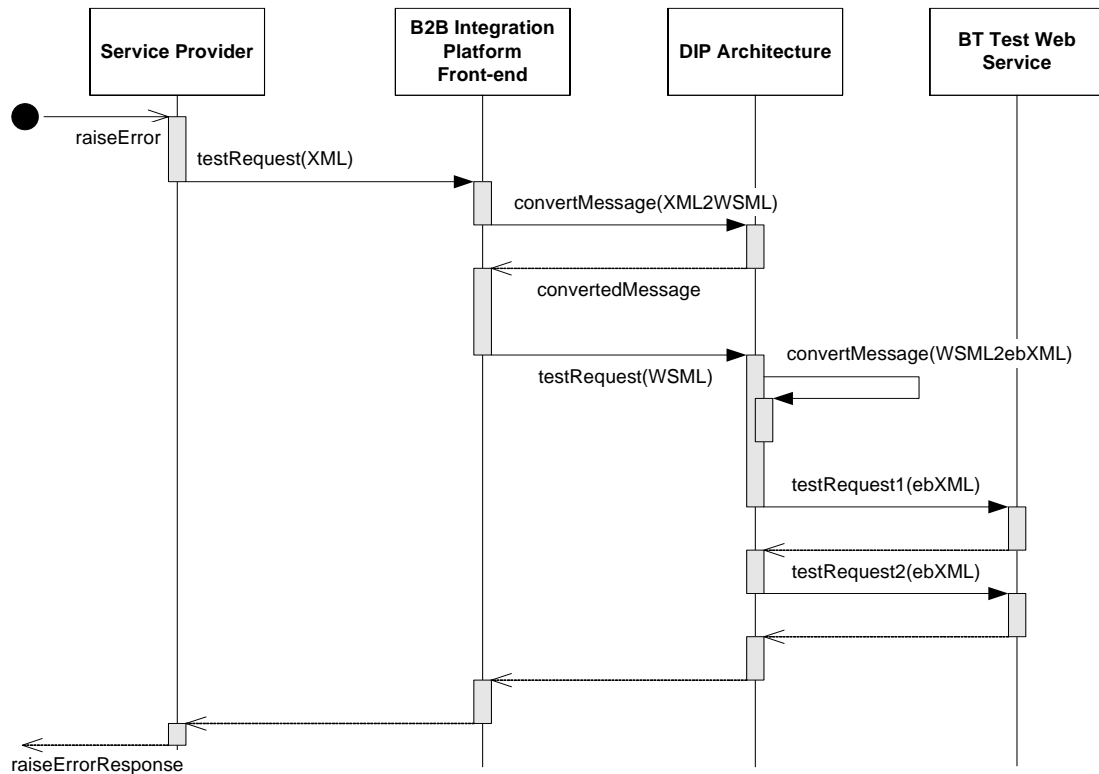


Figure 16: Test Request Sequence Diagram

5.2 Message Conversion

As mentioned in section 2.4, several steps of message conversion are necessary, once for the message sent from the ticketing system, and again before the invocation of the back-end Semantic Web Service. Custom adapters convert the content of a message between the different languages used in the use case, namely: ebXML, the XML dialect used by the Service Provider and WSML.

Scenario description

1. The B2B Integration Platform Front-end requests the conversion of a message specified in one of the possible languages to another one.
2. The DIP Architecture selects a suitable Adapter component, and request a message conversion.
3. The Adapter component returns a message containing the conversion to the B2B Integration platform.

In the particular case of converting from ebXML to the Service Provider's XML format an intermediate conversion to WSML is required.

Sequence Diagram

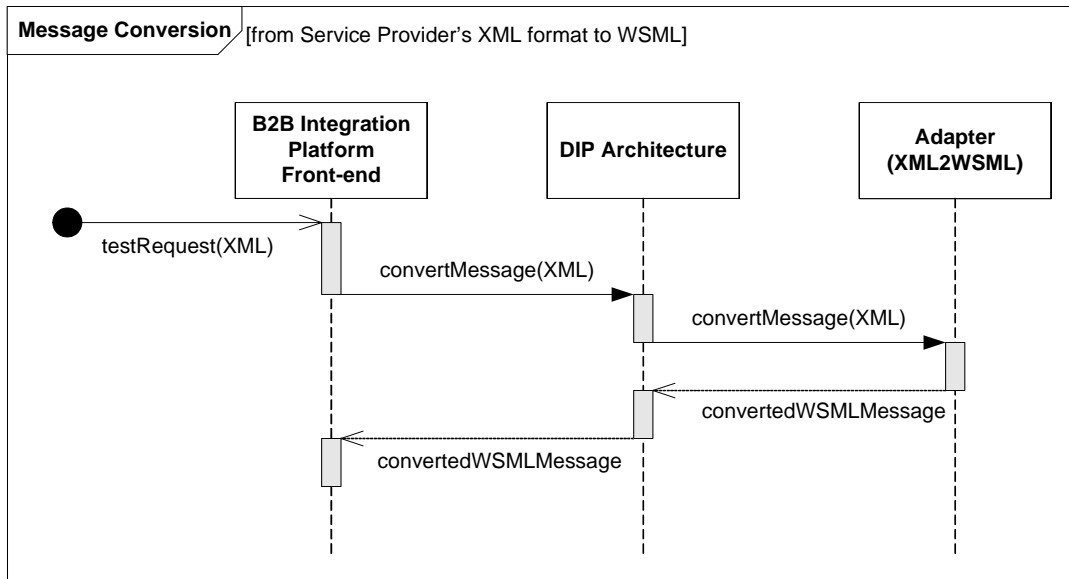


Figure 17: Convert Message (XML2WSML) sequence diagram

All of the necessary message conversion adapters should be built according to a definite set of guidelines, as detailed in section 4.1.2. This is necessary to conform to the requirements set forth by WSMX, to allow the inclusion of the adapter components in the WSMX adapter framework.

5.3 Integration using the DIP API

The DIP project has several workpackages devoted to developing tools for the use of SWS. Many of these tools provide functionality for a specific component of the DIP Architecture. As the architecture aims to be a complete design and runtime solution for SWS it is important that components can communicate with each other in an easy and consistent manner. As part of the goal to provide an 'integrated toolset' a DIP API has been defined. The aim of this was to identify component interfaces and produce a centralised API to describe the main functions of each interface. This will allow component developers to achieve easier integration as well as making the toolset more flexible by allowing third party or additional components to be included if desired.

As consumers of the tools provided by the other workpackages, WP8 will be directly or indirectly using a lot of the APIs defined. As one of the first real testers of some tools it is also likely that experiences gained in deploying the first prototype will have an impact on future versions of the API. As the first prototype will not use all of the functionality defined in the architecture (e.g. Discovery) not the entire API is relevant at this stage. The full definition of the API is available on SourceForge⁷.

Table 6 shows a subset of the top level classes of the API that the prototype will use. The first column gives the name of the Class. The second column is a summary of the features intended for that part of the API. The last column explains how that part of the API will be used in the prototype

⁷ https://sourceforge.net/project/showfiles.php?group_id=113321&package_id=154563

Class	Functionality	Comments
ChoreographyEngine	Handles the message exchange between services based on their defined choreography	This API is not fully defined/implemented at present. For the prototype we will be using the SOPHIE choreography engine. It is anticipated that following the prototype the SOPHIE API will be aligned with this API based on input from WP8
DataMediator	Provides methods for design time mediator definitions and run time mediation.	This is implemented by the WSMX mediation tool being using for the prototype, so will be used indirectly.
OntologyResourceManager	Provides methods for loading and saving ontologies	This will be used by the WSMX mediation tool
WSMORegistry	Provide methods for storing and retrieving WSML messages.	This will be used at design time to store WSML descriptions and also at run time by WSMX to retrieve them.

Table 6 Inputs to *TestRequest* transaction

Any problems or limitations discovered with the APIs or tools implementing them during the development of the prototype will be communicated to relevant parties to ensure that the DIP API provides all the functionality required to achieve the goals of WP8.

5.4 SOPHIE Implementation details

SOPHIE exists as a Web service. The constituent semantic services communicate to realize a complete and coherent implementation of the conceptual model presented. In doing so, it can be readily integrated within the general DIP execution environment, requiring only minimal adaptation effort, centred on the methods that invoke the DIP data mediation component. In the following the topology that will be implemented and the overall algorithm that drives it are presented.

5.4.1 Topologies

Currently, SOPHIE supports the hub and spoke topology.

The *Hub and spoke* topology represents the most evident way of connecting services. As shown in Figure 18, each party is connected to the central correlating service

containing the operational model. The message exchange is not directly between each pair of services, but between each party and the central hub, and vice versa. In a nutshell, the hub service receives messages from either one of the parties. Such messages are mapped and converted to the syntax and semantics expected by the other one, and finally, correlated to the final addressee.

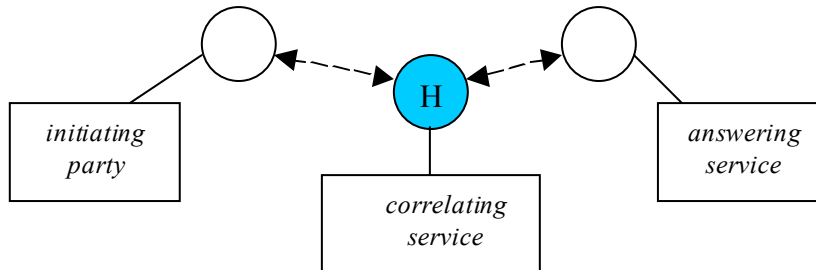


Figure 18: Hub and spoke topology

5.4.2 Overall Algorithm

The overall algorithm allows parties to request the creation of the operational models, correlate messages and finally remove the operational model once it is no longer needed or store it making it available for subsequent re-use.

Given:

- the domain ontologies λ and λ'
- the choreography ontologies θ and θ'
- the input messages δ_i , and δ'_i

Then, the algorithm defines the structures and the logic for the overall functioning of the conceptual model. The pseudocode of the algorithm is detailed in Table 7.

```

1 run ( $\lambda, \lambda', \theta, \theta', \delta_i, \delta'_i$ ):void
2  $T \leftarrow generateOperationalModel(\lambda, \lambda', \theta, \theta')$ 
3 IF messageFromInitiatingParty THEN
    $\tau \leftarrow correlateMessage(T.\tau, \delta_i, \theta')$ 
4 IF messageFromAnsweringService THEN
    $\tau \leftarrow correlateMessage(T.\tau', \delta'_i, \theta)$ 
5 IF removeOperationalModel THEN
   removeOperationalModel ( $\tau$ )

```

Table 7 Pseudocode of the *run* algorithm

In a first step, the operational model that allows the interoperation of syntactically and semantically heterogeneous parties is generated. Then, each time a message is received the defined operational model is used to correlate it according to the choreography of

the addressee. Finally, and in case either party indicates so, the operational model is removed or stored as appropriate.

5.4.3 Integration with the prototype

In order for two parties to communicate, the choreographies for both parties must be known in order to generate a process mediator containing the structures that allow mapping of both the sequence and cardinality of message exchanges. This can be carried out at design time if the two communicating services are known in advance or at run time if not. Such data structures are generated as a result of a reasoning task that invokes the corresponding methods of the DIP execution environment, using the *mediate*⁸ method. They are stored in the correlating service, not requiring the use of additional storage. At run time, every message exchanged among both parties (see sections 2 and 5) will not directly address the recipient, but rather the correlating service, which will take care of converting it to the appropriate format, sending it according to the pattern expected by the addressee.

⁸ org.wsmo.execution.common.component Interface DataMediator

6 Conclusions and Future Work

6.1 Conclusions from this deliverable

This deliverable presents the design for the initial prototype for the Assurance Integration use case. This prototype forms a base line prototype, with an open architecture, which will be expanded to cope with the messages and service requests anticipated in a Telecommunications B2B Integration Platform. This Telecommunications B2B Integration Platform will be a key outcome of DIP Workpackage 8.

This deliverable examines the requirement for design time choreography in detail. The decision to use SOPHIE as the Semantic Choreography has been examined: by using an open architecture, the Choreography engine used in the final version of the DIP Architecture should be suitable for use with this prototype and D8.5.

For the Assurance Integration use case, we have modelled the external visible behaviour of our services with the aim of allowing the interoperation with the platform of services following heterogeneous structures and behaviours. We have also examined what will be available in the delivery window of the prototype deliverable 8.4.

The “Assurance Integration” use case shows a subset of the Telecommunications B2B Integration Platform which may be taken and expanded in deliverable 8.5. The implementation will follow standard engineering practices in producing a DIP based architecture.

The architecture proposed in this deliverable allows the description and use of SWS that will form an integral part of a Semantically Enabled Service Orientated Architecture. The intention is to demonstrate this as a viable approach to producing an extensible prototype in response to the requirements presented in Deliverable 8.2.

6.2 Future Work

The design described in this deliverable will be implemented as a first prototype in D8.4. This will enable adaptation of native messages, data mediation and semantic descriptions of choreography. The prototype will be enhanced as more DIP Architecture components (such as process mediation) become available. The architecture will be extended in a second prototype which will be wider in scope in that it will show the full vision of Semantic Web Services including design and run-time discovery and composition allowing ad hoc, bespoke ICT service provision in-line with the vision identified in D8.2.

Ultimately, this case study aims to prove that SWS are not only a suitable approach to B2B Integration in the Telecommunications sector but the preferable one. The initial approach described in this deliverable and implemented in D8.4 is intended to show the benefits of data and process mediation at the semantic level. As well as providing feedback to the technical workpackages in DIP, it should allow estimates of time and cost savings to be made. D8.5 will be able to demonstrate the advantages of using the semantic web and Semantic Web Services in a large scale environment where the integration of many different parties and services are required and that integration is required with a much reduced level of human intervention.

7 References

1. Zaremba, M., Moran, M. and Hasewanter, T.: “WSMX Architecture”, WSMX Deliverable 13.4 v0.2, available online at: <http://www.wsmo.org/TR/d13/d13.4/v0.2/d13.4.pdf>
2. Booch, G., Rumbaugh, J., and Jacobs, I.: “The Unified Modelling Language User Guide, Addison-Wesley, 1999.
3. Arroyo, S: “SOPHIE - Modeling Choreography: Prospects for Applications to Learning Objects. http://sigrlo.org/newsletter/Arroyo_1_2_Draft_Version.pdf/, April 2005.
4. Duke, A et al.: “D8.2: Platform Requirements Specification”, DIP Project, WP8 B2B in Telecommunications. <http://dip.semanticweb.org>.
5. Sapkota, B: “WSMX Adaptor Framework”, WSMX deliverable in preparation (<http://www.wsmo.org/wsmx/publications.html>).
6. Moran, M, and Mocan, A: “Towards Translating between XML and WSML based on mappings between XML Schema and an equivalent WSMO Ontology”, 2nd WSMO Implementation Workshop (WIW), 2005, <http://www.wsmo.org/wiw/2005/>
7. Cimpian E, Zaremba, M, et al, DERI Galway: “Web Service Execution Environment (WSMX)”, submission to W3C

Appendix 1: WSMO Descriptions

The appendix provides the domain ontology and WSMO descriptions of the services provided in the prototype being produced in D8.4.

The Domain Ontology

```
namespace <<http://localhost/ontologies/eCo>>
    dc:<<http://purl.org/dc/elements/1.1#>>
    dt:<<http://www.wsmo.org/ontologies/dateTime#>>
    xsd:<<http://www.w3.org/2001/XMLSchema#>>
    targetnamespace: <<http://localhost/ontologies/eCo#>>

ontology <<http://localhost/ontologies/eCo>>
nonFunctionalProperties
    dc:description hasValue "This Ontolgy describes the main concepts of the
    BT eCo Assurance Integration gateway. The gateway allows customers of BTs
    wholesale services to access assurance funtionaility, such as the ability
    to request a test on the Network if a fault has occured"
endNonFunctionalProperties

Concept GatewayUser
    PartyID      ofType XSD:String
    AgencyID     of Type XSD:String
    nonFunctionalProperties
        dc:description hasValue "Top level Concept for all eCo Gateway
Users"
    endNonFunctionalProperties

Concept Performer SubConceptOf GatewayUser
    nonFunctionalProperties
        dc:description hasValue "Gatway User performing a request"
    endNonFunctionalProperties

Concept Maintainer SubConceptOf GatewayUser
    nonFunctionalProperties
        dc:description hasValue "Gateway user responsible for maintenance"
    endNonFunctionalProperties

Concept Conductor SubConceptOf GatewayUser
    ContactName  ofType Fullname
    Telephone    ofType InternationalPhoneNumber
    FirstName    ofType Forname
```

```

nonFunctionalProperties
    dc:description hasValue "Gateway user conducting specific Tests"
endNonFunctionalProperties

Concept Reporter SubConceptOf GatewayUser
    ContactName    ofType Fullname
    Telephone      ofType InternationalPhoneNumber
    FirstName      ofType Forname
    nonFunctionalProperties
        dc:description hasValue "Gateway user reporting a Fault"
    endNonFunctionalProperties

Concept Reference
    ReferenceNumber ofType XSD:Integer
    nonFunctionalProperties
        dc:description hasValue "Top level concept for all Reference"
    endNonFunctionalProperties

Concept TestReference subConceptOf Reference
    TestDate      ofType dt:Date
    nonFunctionalProperties
        dc:description hasValue "Referrnace for tests"
    endNonFunctionalProperties

Concept TroubleReportingReference subConceptOf Reference
    nonFunctionalProperties
        dc:description hasValue "Reference for trouble reports"
    endNonFunctionalProperties

Concept TroubleMaintenanceReference subConceptOf Reference
    nonFunctionalProperties
        dc:description hasValue "Reference for maintainer"
    endNonFunctionalProperties

Concept TroubleReportMessageInfo
    Code ofType XSD:String
    nonFunctionalProperties
        dc:description hasValue "Status of Trouble Report"
    endNonFunctionalProperties

```

```

Concept TroubleNotificationMessageInfo
    Code ofType XSD:String
    Message ofType XSD:String
    nonFunctionalProperties
        dc:description hasValue "Status of Trouble Report"
    endNonFunctionalProperties

Concpet TroubleReportDetail
    Description ofType XSD:String
    AgencyID ofType XSD:String
    FaultCode ofType XSD:String
    TroubleServiceQuestions ofType eCo:ServiceQuestions
    ServiceLevel ofType XSD:String
    Note ofType XSD:String
    nonFunctionalProperties
        dc:description hasValue "Details of Trouble Report"
    endNonFunctionalProperties

Concept ServiceQuestions
    ServiceHasWorked ofType XSD:boolean
    EquipmentChecked ofType XSD:boolean
    ServiceCurrentlyInUse ofType XSD:boolean
    DisruptiveTestsAllowed ofType XSD:boolean
    PowerChecked ofType XSD:boolean
    nonFunctionalProperties
        dc:description hasValue "Set of Questions required before Trouble
Test can be carried out"
    endNonFunctionalProperties

Concept InvokationIdentifer
    RefNumber ofType XSD:String
    nonFunctionalProperties
        dc:description hasValue "Top Level Concept for Invokation
Identifier"
    endNonFunctionalProperties

Concept PerformerInvokationIdentifer SubConceptOf InvokationIdentifer
    nonFunctionalProperties
        dc:description hasValue "Invokation Identifier for Performer"
    endNonFunctionalProperties

```

```
Concept ConductorInvokationIdentifer SubConceptOf InvokationIdentifer
  nonFunctionalProperties
    dc:description hasValue "Invokation Identifier for Conductor"
  endNonFunctionalProperties
```

```
Concept TestCategory
  Category ofType {IncidentCheck, oneShotCheck, XLMSCheck}
  nonFunctionalProperties
    dc:description hasValue "Type of Test to be carried out, e.g
OneShot"
  endNonFunctionalProperties
```

```
Concept Status
  Status ofType {Accepted,Rejected}
  nonFunctionalProperties
    dc:description hasValue "Status of a Request"
  endNonFunctionalProperties
```

```
Concept TestStatus subConceptOf Status
  nonFunctionalProperties
    dc:description hasValue "Status of a Test Request"
  endNonFunctionalProperties
```

```
Concept AppointmentStatus subConceptOf Status
  nonFunctionalProperties
    dc:description hasValue "Status of an Appointment Request"
  endNonFunctionalProperties
```

```
Concept TroubleReportStatus subConceptOf Status
  nonFunctionalProperties
    dc:description hasValue "Status of a Trouble Report"
  endNonFunctionalProperties
```

```
Concept ErrorMessage
  Code ofType XSD:Integer
  Message ofType XSD:String
  nonFunctionalProperties
    dc:description hasValue "Error Message and Code"
```

endNonFunctionalProperties

Concept Incident

Details ofType XSD:String
StartDate ofType dt:Date
EndDate ofType dt:Date
EstimatedCompletionDate ofType dt:Date
Duration ofType dt:hours
Status ofType XSD:String
nonFunctionalProperties
 dc:description hasValue "Details of an Incident"
endNonFunctionalProperties

Concept TestOutcome

ActualStartTime ofType dt:Date
TestResultDate ofType dt:Date
TestCategory ofType testCategory (oneShot, XLMS etc..)
Outcome ofType ofType XSD:String
Summary ofType ofType XSD:String
AdditionalText ofType XSD:String
nonFunctionalProperties
 dc:description hasValue "Results of a Tests"
endNonFunctionalProperties

Concept AppointmentRequest subConceptOf dt:Date

nonFunctionalProperties
 dc:description hasValue "Date when an Appointment is required"
endNonFunctionalProperties

Concept AlternativeAppointmentDate

AlternativeAppointmentDate ofType dt:Date
nonFunctionalProperties
 dc:description hasValue "Date when appointment is available"
endNonFunctionalProperties

Concept AccessDetails

Details ofType XSD:String
nonFunctionalProperties
 dc:description hasValue "Access details for engineer onto premises"
endNonFunctionalProperties

Concept Appointment

Date ofType dt:Date

StartTime ofType dt:Time

EndTime ofType dt:Time

nonFunctionalProperties

dc:description hasValue "Date and time slot for an engineer appointment"

endNonFunctionalProperties

Concept WorldState

State ofType {Ready, Awaiting, Completed, Failed}

nonFunctionalProperties

dc:description hasValue "Describes Real World status of a Request"

endNonFunctionalProperties

The WSMML Descriptions for the defined services

RequestTest

```
namespace:    <<http://localhost/Assurance#>>
dc:           <<http://purl.org/dc/elements/1.1#>>
dt:           <<http://www.wsmo.org/ontologies/dateTime#>>
eCo:         <<http://localhost/ontologies/eCo#>>
eTom:        <<http://localhost/ontologies/eTom#>>
sid:         <<http://localhost/ontologies/sid#>>

nonFunctionalProperties
    dc:title hasValue "Request Incident Check Web Service"
    dc:creator hasValue "Marc Richardson"
    dc:subject hasValue "eTOM:EvaluateAndQualifyProblem"
    dc:description hasValue "Will Initiate a request for an incident
check on the BT eCo platform"
    dc:publisher hasValue "British Telecommunications PLC"
    dc:contributor hasValues {"Marc Richardson"}
    dc:date hasValue "2004-12-15"
    dc:type hasValue <<http://www.wsmo.org/2004/d2/#webservice>>
    dc:format hasValue "text/html"
    dc:language hasValue "en-us"
    dc:rights hasValue <<http://www.bt.com/privacy.html>>
    version hasValue "$Revision: 1.0 $"
endNonFunctionalProperties

webservice Test

capability RequestTest

preCondition
    axiom InputsRequired
    nonFunctionalProperties
        dc:description hasValue "The preconditions state the inputs that
are required from the webservice
        and that the Test Catagory must be of one of the three valid tests
"
    endNonFunctionalProperties
    definedBy
        ?inputTestReferance memberOf eCo:TestReferance[
            eCo:TestDate          hasValue ?inputTestDate
```

```

        eCo:ReferenceNumber      hasValue ?inputReferenceNumber
    ] and

    ?inputPerformer memberOf eCo:Performer[
        eCo:PartyID hasValue      ?inputPerformerPartyID
        eCo:AgencyID has Value    ?inputPerformerAgencyID
    ] and

    ?inputConductor memberOf eCo:Condcutor [
        eCo:PartyID hasValue ?inputConductorPartyID
        eCo:AgencyID hasValue ?inputConductorAgencyID
        eCo:ContactName hasValue ?inputContactName
        eCo:Telephone      hasValue ?inputTelephone
        eCo:FirstNamehasValue ?inputFirstName
    ] and

    ?inputTestCategory memberOf eCo:TestCatagory [
        eCo:catagory hasValue ?inputTestCategory
    ] and ?inputTestCategory hasValue "IncidentCheck" or hasValue
    "oneShotCheck" or hasValue "XLMSCheck"

assumption
    axiom InitialRequestState
    nonFunctionalProperties
        dc:description hasValue "Describes the status of the request
before processing"
    endNonFunctionalProperties
    definedBy

    ?StatingState memberOf WorldState [
        eCo:state hasValue "Ready"
    ]

postCondition
    axiom OutputsReturned
    nonFunctionalProperties
        dc:description hasValue "The Service will return all of the
information presented as inputs+

```

test status message. If the test status is 'rejected' then the test was not carried out and

and error code/message is returned. If the test status is 'accepted' then the test result information is

returned"

endNonFunctionalProperties

definedBy

```
?inputTestReferance memberOf eCo:TestReferance[
    eCo:TestDate      hasValue ?inputTestDate
    eCo:ReferenceNumber  hasValue ?inputReferenceNumber
] and
```

```
?inputPerformer memberOf eCo:Performer[
    eCo:PartyID  hasValue ?inputPerformerPartyID
    eCo:AgencyID hasValue ?inputPerformerAgencyID
] and
```

```
?inputConductor memberOf eCo:Condcutor [
    eCo:PartyID  hasValue ?inputConductorPartyID
    eCo:AgencyID hasValue ?inputConductorAgencyID
    eCo:ContactName hasValue ?inputContactName
    eCo:Telephone  hasValue ?inputTelephone
    eCo:FirstNamehasValue ?inputFirstName
] and
```

```
?inputTestCategory memberOf eCo:TestCatagory [
    eCo:catagory hasValue ?inputTestCategory
] and
```

(?inputTestCategory hasValue "IncidentCheck" or hasValue "oneShotCheck" or hasValue "XLMSCheck")

and ((

```
?outputTestStatus memberOf TestStatus [
    ecoTestStatus  hasValue "rejected"
] and
```

```
?outputErrorMessage memberOf ErrorMessage [
    eCo:code      hasValue ?outputCode
    eCo:message   hasValue ?outputMessage
```

```

]

) or (

?outputTestStatus memberOf TestStatus [
    ecoTestStatus      hasValue "accepted"
] and

?outputIncident memberOf Incident [
    eco:details      hasValue ?outputDetails
    eco:startDate    hasValue ?outputStartDate
    eco:endDate      hasValue ?outputEndDate
    eco:EstimatedCompletionDate hasValue ?outputEstimatedCompletionDate
    eco:Duration     hasValue ?outputDuration
    eco:Status       hasValue ?outputStatus
]and

Concept ?outputTestOutcome memberOf TestOutcome [
    eco:ActualStartTime hasValue ?outputActualStartTime
    eco:TestResultDate  hasValue ?outputTestResultDate
    eco:TestCategory    hasValue ?outputTestCategory
    eco:Outcome ofType  hasValue ?outputOutcome
    eco:Summary ofType  hasValue ?outputSummary
    eco:AdditionalText  hasValue ?outputAdditionalText
]and

?outputPerformerInvokationIdentifer memberOf PerformerInvokationIdentifer
[
    eco:RefNumber hasValue PerformerInvokationRefNumber
]and

?outputConductorInvokationIdentifer memberOf ConductorInvokationIdentifer
[
    eco:RefNumber hasValue ConductorInvokationRefNumber
]
))

effect
axiom EndRequestState
nonFunctionalProperties

```

```

        dc:description hasValue "Describes the status of the request
After processing"
    endNonFunctionalProperties

    ?EndState memberOf WorldState [
        eCo:state hasValue {Completed,Failed}
    ]

```

RequestTroubleTicket

```

namespace:    <<http://localhost/Assurance#>>
dc:           <<http://purl.org/dc/elements/1.1#>>
dt:          <<http://www.wsmo.org/ontologies/dateTime#>>
eCo:         <<http://localhost/ontologies/eCo#>>
eTom:        <<http://localhost/ontologies/eTom#>>
sid:         <<http://localhost/ontologies/sid#>>

nonFunctionalProperties
    dc:title hasValue "Request Trouble Report Web Service"
    dc:creator hasValue "Marc Richardson"
    dc:subject hasValue "eTOM:EvaluateAndQualifyProblem"
    dc:description hasValue "Will Initiate a request for a Trouble
Report on the BT eCo platform"
    dc:publisher hasValue "British Telecommunications PLC"
    dc:contributor hasValues {"Marc Richardson"}
    dc:date hasValue "2004-12-15"
    dc:type hasValue <<http://www.wsmo.org/2004/d2/#webservice>>
    dc:format hasValue "text/html"
    dc:language hasValue "en-us"
    dc:rights hasValue <<http://www.bt.com/privacy.html>>
    version hasValue "$Revision: 1.0 $"
endNonFunctionalProperties

webservice TroubleReport

capability RequestTroubleReport

precondition
    axiom InputsRequired
    nonFunctionalProperties
        dc:description hasValue "The preconditions state the inputs
that are required from the webservice"

```

```

endNonFunctionalProperties
definedBy
?inputTestReferance memberOf eCo:TestReferance[
    eCo:ReferenceNumber      hasValue ?inputReferenceNumber
] and

?inputReportingParty memberOf eCo:Reporter[
    eCo:PartyID      hasValue ?inputReporterPartyID
    eCo:AgencyID    hasValue ?inputReporterAgencyID
    eCo:ContactName  hasValue ?inputContactName
    eCo:Telephone     hasValue ?inputTelephone
    eCo:FirstNamehasValue ?inputFirstName
] and

?inputMaintainer memberOf eCo:Maintainer [
    eCo:PartyID      hasValue      ?inputMaintainerPartyID
    eCo:AgencyID    hasValue      ?inputMaintainerAgencyID
] and

?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo[
    eCo:code hasValue ?inputcode
] and

?inputTroubleReportDetail memberOf eCo:TroubleReportDetail [
    eCo:Description      hasValue ?inputDescription
    eCo:AgencyID        hasValue ?inputReporterAgencyID
    eCo:FaultCode        hasValue ?inputFaultCode
    eCo:TroubleServiceQuestions      hasValue
?inputTroubleServiceQuestions
    eCo:ServiceLevel    hasValue ?inputServiceLevel
    eCo:Note             hasValue ?inputNote
]

assumption
    axiom InitialRequestState
    nonFunctionalProperties
        dc:description hasValue "Describes the status of the request
before processing"
    endNonFunctionalProperties
definedBy
?StatingState memberOf WorldState [

```

```

        eCo:state hasValue "Ready"
    ]

postCondition
    axiom OutputsReturned

    nonFunctionalProperties
        dc:description hasValue "The Service will return all of the
information presented as inputs+
        Trouble report status message. If the test status is 'rejected'
then the test was not carried out and
        and error code/message is returned. If the test status is
'accepted' then the Trouble report result information is
        returned"
    endNonFunctionalProperties
    definedBy
    ?inputTestReferance memberOf eCo:TestReferance[
        eCo:ReferenceNumber      hasValue ?inputReferenceNumber
    ] and

    ?inputReportingParty memberOf eCo:Reporter[
        eCo:PartyID      hasValue ?inputReporterPartyID
        eCo:AgencyID hasValue ?inputReporterAgencyID
        eCo:ContactName hasValue ?inputContactName
        eCo:Telephone      hasValue ?inputTelephone
        eCo:FirstNamehasValue ?inputFirstName
    ] and

    ?inputMaintainer memberOf eCo:Maintainer [
        eCo:PartyID hasValue      ?inputMaintainerPartyID
        eCo:AgencyID hasValue      ?inputMaintainerAgencyID
    ] and

    ?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo[
        eCo:code hasValue ?inputcode
    ] and

    ?inputTroubleReportDetail memberOf eCo:TroubleReportDetail [
        eCo:Description      hasValue ?inputDescription
        eCo:AgencyID      hasValue ?inputReporterAgencyID
        eCo:FaultCode      hasValue ?inputFaultCode
    ]

```

```

                eCo:TroubleServiceQuestions      hasValue
?inputTroubleServiceQuestions
                eCo:ServiceLevel                hasValue ?inputServiceLevel
                eCo:Note                        hasValue ?inputNote
    ] and ((

        ?outputTroubleMaintenanceReferance      memberOf
eCo:TroubleMaintenanceReferance [
                ReferenceNumber ofType XSD:Integer
    ] and

        ?outputErrorMessage memberOf eCo:ErrorMessage [
                eCo:code      hasValue ?outputCode
                eCo:message   hasValue ?outputMessage
        ] ) or (

        ?outputTroubleMaintenanceReferance      memberOf
eCo:TroubleMaintenanceReferance [
                eCo:ReferenceNumber hasValue ?outputReferenceNumber
    ] and

        ?outputTroubleNotificationMessageInfo   memberOf
eCo:TroubleNotificationMessageInfo
                eCo:code      hasValue ?outputCode
                eCo:message   hasValue ?outputMessage
    ))

effect
    axiom EndRequestState
    nonFunctionalProperties
        dc:description hasValue "Describes the status of the request
After processing"
    endNonFunctionalProperties
    definedBy

    ?EndState memberOf WorldState [
        eCo:state hasValue {Completed,Failed}
    ]

```

RequestAppointmentAvailability

```
namespace: <<http://localhost/Assurance#>>
dc:        <<http://purl.org/dc/elements/1.1#>>
dt:        <<http://www.wsmo.org/ontologies/dateTime#>>
eCo:       <<http://localhost/ontologies/eCo#>>
eTom:      <<http://localhost/ontologies/eTom#>>
sid:       <<http://localhost/ontologies/sid#>>
```

webservice BroadBandRepair

capability RequestAppointmentAvailability

nonFunctionalProperties

```
    dc:title    hasValue    "BroadBand Repair, Request Appointment
Availability Web Service"
    dc:creator  hasValue    "Marc Richardson"
    dc:subject  hasValue    "eTOM:EvaluateAndQualifyProblem"
    dc:description hasValue "Will Request all BroadBand Repair
appointments available for a specific date on BT eCo platform"
    dc:publisher hasValue "British Telecommunications PLC"
    dc:contributor hasValues {"Marc Richardson"}
    dc:date     hasValue    "2004-12-15"
    dc:type     hasValue    <<http://www.wsmo.org/2004/d2/#webservice>>
    dc:format   hasValue    "text/html"
    dc:language hasValue    "en-us"
    dc:rights   hasValue    <<http://www.bt.com/privacy.html>>
    version    hasValue    "$Revision: 1.0 $"
```

endNonFunctionalProperties

preCondition

```
    axiom InputsRequired
    definedBy
    ?inputTestReferance memberOf eCo:TestReferance[
        eCo:ReferenceNumber    hasValue ?inputReferenceNumber
    ] and

    ?inputReportingParty memberOf eCo:Reporter[
        eCo:PartyID    hasValue ?inputReporterPartyID
        eCo:AgencyID  hasValue ?inputReporterAgencyID
        eCo:ContactName hasValue ?inputContactName
```

```

        eCo:Telephone      hasValue ?inputTelephone
        eCo:FirstNamehasValue ?inputFirstName
    ] and

    ?inputMaintainer memberOf eCo:Maintainer [
        eCo:PartyID  hasValue ?inputMaintainerPartyID
        eCo:AgencyID hasValue ?inputMaintainerAgencyID
    ] and

    ?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo[
        eCo:code      hasValue ?inputCode
    ] and

    ?inputAppointmentRequest memberOf eCo:Appointment[
        dt:date      hasValue ?inputDate
    ]

assumption
    axiom InitialRequestState

    nonFunctionalProperties
        dc:description hasValue "Describes the status of the request
before processing"
    endNonFunctionalProperties
    definedBy
    ?StatingState memberOf WorldState [
        eCo:state hasValue "Ready"
    ]

postCondition
    axiom OutputsReturned
    nonFunctionalProperties
        dc:description hasValue "The Service will return all of the
information presented as inputs+
        a set of Availible Appointrment slots. If request status is
'rejected' an error code/message is returned.
    endNonFunctionalProperties
    definedBy
    ?inputTestReferance memberOf eCo:TestReferance[
        eCo:ReferenceNumber      hasValue ?inputReferenceNumber
    ] and

```

```

?inputReportingParty memberOf eCo:Reporter[
    eCo:PartyID hasValue ?inputReporterPartyID
    eCo:AgencyID hasValue ?inputReporterAgencyID
    eCo:ContactName hasValue ?inputContactName
    eCo:Telephone hasValue ?inputTelephone
    eCo:FirstName hasValue ?inputFirstName
] and

?inputMaintainer memberOf eCo:Maintainer [
    eCo:PartyID hasValue ?inputMaintainerPartyID
    eCo:AgencyID hasValue ?inputMaintainerAgencyID
] and

?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo[
    eCo:code hasValue ?inputcode
] and ((

/**Array Of**/
?outputAppointment memberOf eCo:Appointment [
    eCo:Date hasValue ?outputDate
    eCo:StartTime hasValue ?outputStartTime
    eCo:EndTime hasValue ?outputEndTime
]

/**end Array of**/

) or (

?outputErrorMessage memberOf eCo:ErrorMessage [
    eCo:code hasValue ?outputCode
    eCo:message hasValue ?outputMessage
]
))

```

effect

```

axiom EndRequestState
nonFunctionalProperties

```

```

                dc:description hasValue "Describes the status of the request
After processing"
            endNonFunctionalProperties
            definedBy

            ?EndState memberOf WorldState [
                eCo:state hasValue {Completed,Failed}
            ]

```

MakeAppointment

```

namespace:    <<http://localhost/Assurance#>>
dc:           <<http://purl.org/dc/elements/1.1#>>
dt:          <<http://www.wsmo.org/ontologies/dateTime#>>
eCo:         <<http://localhost/ontologies/eCo#>>
eTom:        <<http://localhost/ontologies/eTom#>>
sid:         <<http://localhost/ontologies/sid#>>

```

webservice BroadBandRepair

capability MakeAppointment

nonFunctionalProperties

```

                dc:title hasValue "BroadBand Repair, Request Appointment
Availability Web Service"
                dc:creator hasValue "Marc Richardson"
                dc:subject hasValue "eTOM:EvaluateAndQualifyProblem"
                dc:description hasValue "Will Request to make a BroadBand Repair
appointment for a specific date on BT eCo platform"
                dc:publisher hasValue "British Telecommunications PLC"
                dc:contributor hasValues {"Marc Richardson"}
                dc:date hasValue "2004-12-15"
                dc:type hasValue <<http://www.wsmo.org/2004/d2/#webservice>>
                dc:format hasValue "text/html"
                dc:language hasValue "en-us"
                dc:rights hasValue <<http://www.bt.com/privacy.html>>
                version hasValue "$Revision: 1.0 $"

```

endNonFunctionalProperties

preCondition

```

    axiom InputsRequired
    definedBy

```

```

?inputTestReferance memberOf eCo:TestReferance[
    eCo:ReferenceNumber      hasValue ?inputReferenceNumber
] and

?inputReportingParty memberOf eCo:Reporter[
    eCo:PartyID      hasValue ?inputReporterPartyID
    eCo:AgencyID    hasValue ?inputReporterAgencyID
    eCo:ContactName  hasValue ?inputContactName
    eCo:Telephone     hasValue ?inputTelephone
    eCo:FirstNamehasValue ?inputFirstName
] and

?inputMaintainer memberOf eCo:Maintainer [
    eCo:PartyID      hasValue ?inputMaintainerPartyID
    eCo:AgencyID    hasValue ?inputMaintainerAgencyID
] and

?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo[
    eCo:code         hasValue ?inputCode
] and

?inputApointment memberOf eCo:Apointment [
    eCo:Date         hasValue ?inputDate
    eCo:StartTime    hasValue ?inputStartTime
    eCo:EndTime      hasValue ?inputEndTime
] and

?inputAlternativeAppointmentDate memberOf eCo:AlternativeAppointmentDate
[
    eCo:Date         hasValue ?inputAlternativeAppointmentDate
]

assumption
    axiom InitialRequestState
    nonFunctionalProperties
        dc:description hasValue "Describes the status of the request
before processing"
    endNonFunctionalProperties
    definedBy
    ?StatingState memberOf WorldState [

```

```

        eCo:state hasValue "Ready"
    ]

postCondition
    axiom OutputsReturned
    nonFunctionalProperties
        dc:description hasValue "The Service will book and appointment for
a Broadband repair."
    endNonFunctionalProperties
    definedBy
        ?inputTestReferance memberOf eCo:TestReferance[
            eCo:ReferenceNumber      hasValue ?inputReferenceNumber
        ] and

        ?inputReportingParty memberOf eCo:Reporter[
            eCo:PartyID      hasValue ?inputReporterPartyID
            eCo:AgencyID    hasValue ?inputReporterAgencyID
            eCo:ContactName  hasValue ?inputContactName
            eCo:Telephone     hasValue ?inputTelephone
            eCo:FirstNamehasValue ?inputFirstName
        ] and

        ?inputMaintainer memberOf eCo:Maintainer [
            eCo:PartyID      hasValue ?inputMaintainerPartyID
            eCo:AgencyID    hasValue ?inputMaintainerAgencyID
        ] and

        ?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo[
            eCo:code         hasValue ?inputcode
        ] and ((

        ?outputApoinmentStatus memberof eCo:AppointmentStatus [
            eCo:Status hasValue "accepted"
        ] and

        ?outputApoinment memberOf eCo:Appointment [
            eCo:Date      hasValue ?inputputDate
            eCo:StartTime      hasValue ?inputputStartTime
            eCo:EndTime  hasValue ?inputputEndTime
        ]
    ]

```

```

) or (

/** Array of */

?outputAppointment memberOf eCo:Appointment [
    eCo:Date      hasValue ?outputDate
    eCo:StartTime  hasValue ?outputStartTime
    eCo:EndTime   hasValue ?outputEndTime
]

/** Array of */

) or (

?outputAppointmentStatus memberOf eCo:AppointmentStatus [
    eCo:Status hasValue "rejected"
] and
?outputErrorMessage memberOf eCo:ErrorMessage [
    eCo:code      hasValue ?outputCode
    eCo:message   hasValue ?outputMessage
]
)))

effect
    axiom EndRequestState
    nonFunctionalProperties
        dc:description hasValue "Describes the status of the request
After processing"
    endNonFunctionalProperties
    definedBy

    ?EndState memberOf WorldState [
        eCo:state hasValue {Completed,Failed}
    ]

```

RequestAppointmentCancellation

```

namespace: <<http://localhost/Assurance#>>
dc:        <<http://purl.org/dc/elements/1.1#>>
dt:        <<http://www.wsmo.org/ontologies/dateTime#>>
eCo:       <<http://localhost/ontologies/eCo#>>

```

eTom: <<http://localhost/ontologies/eTom#>>
sid: <<http://localhost/ontologies/sid#>>

webservice BroadBandRepair

capability RequestAppointmentCancellation

nonFunctionalProperties

dc:title hasValue "BroadBand Repair, Request Appointment
Cancellation Web Service"

dc:creator hasValue "Marc Richardson"

dc:subject hasValue "eTOM:EvaluateAndQualifyProblem"

dc:description hasValue "Will Request a Cancellation of BroadBand
Repair appointments on BT eCo platform"

dc:publisher hasValue "British Telecommunications PLC"

dc:contributor hasValues {"Marc Richardson"}

dc:date hasValue "2004-12-15"

dc:type hasValue <<http://www.wsmo.org/2004/d2/#webservice>>

dc:format hasValue "text/html"

dc:language hasValue "en-us"

dc:rights hasValue <<http://www.bt.com/privacy.html>>

version hasValue "\$Revision: 1.0 \$"

endNonFunctionalProperties

preCondition

axiom InputsRequired

nonFunctionalProperties

dc:description hasValue "The preconditions state the inputs
that are required from the webservice"

endNonFunctionalProperties

definedBy

?inputTestReferance memberOf eCo:TestReferance[

eCo:ReferenceNumber hasValue ?inputReferenceNumber

] and

?inputReportingParty memberOf eCo:Reporter[

eCo:PartyID hasValue ?inputReporterPartyID

eCo:AgencyID hasValue ?inputReporterAgencyID

eCo:ContactName hasValue ?inputContactName

eCo:Telephone hasValue ?inputTelephone

eCo:FirstNamehasValue ?inputFirstName

] and

```
?inputMaintainer memberOf eCo:Maintainer [  
    eCo:PartyID hasValue ?inputMaintainerPartyID  
    eCo:AgencyID hasValue ?inputMaintainerAgencyID  
]
```

] and

```
?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo [  
    eCo:code hasValue ?inputCode  
]
```

] and

```
?inputAppointment memberOf eCo:Appointment [  
    eCo:Date hasValue ?inputDate  
    eCo:StartTime hasValue ?inputStartTime  
    eCo:EndTime hasValue ?inputEndTime  
]
```

]

assumption

```
axiom InitialRequestState  
definedBy  
nonFunctionalProperties  
    dc:description hasValue "Describes the status of the request  
before processing"  
endNonFunctionalProperties
```

```
?StatingState memberOf WorldState [  
    eCo:state hasValue "Ready"  
]
```

]

postCondition

```
axiom OutputsReturned  
nonFunctionalProperties  
    dc:description hasValue "The Service will either cancel the  
appointment or return an error message"  
endNonFunctionalProperties
```

definedBy

```
?inputReportingParty memberOf eCo:Reporter [  
    eCo:PartyID hasValue ?inputReporterPartyID  
    eCo:AgencyID hasValue ?inputReporterAgencyID  
    eCo:ContactName hasValue ?inputContactName  
    eCo:Telephone hasValue ?inputTelephone  
]
```

```

        eCo:FirstNamehasValue ?inputFirstName
    ] and

    ?inputMaintainer memberOf eCo:Maintainer [
        eCo:PartyID hasValue ?inputMaintainerPartyID
        eCo:AgencyID hasValue ?inputMaintainerAgencyID
    ] and

    ((

    ?inputTroubleReportMessageInfo memberOf eCo:TroubleReportMessageInfo[
        eCo:code hasValue ?outputCode
    ]

    ) or (

    ?outputErrorMessage memberOf eCo:ErrorMessage [
        eCo:code hasValue ?outputCode
        eCo:message hasValue ?outputMessage
    ]

    ))

effect
    axiom EndRequestState
    nonFunctionalProperties
        dc:description hasValue "Describes the status of the request
After processing"
    endNonFunctionalProperties
    definedBy
    ?EndState memberOf WorldState [
        eCo:state hasValue {Completed,Failed}
    ]

```

Appendix 2: Conceptual Choreography Ontology

This appendix details the ontology of the conceptual model, that can be used to model the MEPs of services.

```
namespace {<"http://www.deri.org/ontology/choreographyEngine">,  
    dc <"http://purl.org/dc/elements/1.1#">,  
    xsd <"http://www.w3.org/2001/XMLSchema#">,  
    }ontology <"http://www.deri.org/ontology/choreographyOntology#">
```

nonFunctionalProperties

```
dc#title hasValue "Choreography Ontology"  
dc#creator hasValue "Sinuhe Arroyo"  
    dc#description hasValue "an ontology for describing the concepts  
    of the choreography engine"  
dc#publisher hasValue "DERI International"  
dc#contributor hasValues "Oze"  
dc#date hasValue "2005-04-11"  
dc#type hasValue <"http://www.deri.org/2005/#ontology">  
dc#format hasValue "text/html"  
dc#language hasValue "en-us"  
dc#rights hasValue <"http://deri.at/privacy.html">  
version hasValue "$Revision 0.1$"
```

endNonFunctionalProperties

concept entity

nonFunctionalProperties

```
dc#description hasValue "entities of the choreography engine "
```

endNonFunctionalProperties

concept identifier

nonFunctionalProperties

```
dc#description hasValue "key value to designate entities"
```

endNonFunctionalProperties

concept handler

nonFunctionalProperties

dc#description hasValue "address and protocol of a concrete entity"

endNonFunctionalProperties

concept uri

nonFunctionalProperties

dc#description hasValue "handler, entity and identifier"

endNonFunctionalProperties

hasHandler ofType (1 1) handler

hasEntity ofType (1 1) entity

hasIdentifier ofType (1 1) identifier

concept name

nonFunctionalProperties

dc#description hasValue "identifies an entity"

endNonFunctionalProperties

concept role

nonFunctionalProperties

*dc#description hasValue "specifies whether a party is an initiating party
or an answering service"*

endNonFunctionalProperties

concept party

nonFunctionalProperties

*dc#description hasValue "active entities that interact with the
choreography engine"*

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasRole ofType (1 1) role

concept type

nonFunctionalProperties

dc#description hasValue "XSD type of the element"

endNonFunctionalProperties

concept value

nonFunctionalProperties

dc#description hasValue "value of the element"

endNonFunctionalProperties

concept element

nonFunctionalProperties

dc#description hasValue "piece of data either supplied or consumed by parties"

endNonFunctionalProperties

hasName ofType (1 1) name

hasType ofType (1 1) type

hasValue ofType (1 1) value

concept businessRule

nonFunctionalProperties

dc#description hasValue "define additional relevant information over elements in documents"

endNonFunctionalProperties

concept document

nonFunctionalProperties

dc#description hasValue "grouping of a number related elements"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

*hasElements ofType (0 *) element*

*hasBusinessRules ofType (1 *) businessRule*

concept *messge*

nonFunctionalProperties

dc#description hasValue "minimal unit that can be exchanged among parties"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasFrom ofType (0 1) party

hasTo ofType (0 1) party

*hasDocuments ofType (0 *) document*

concept *mep*

nonFunctionalProperties

dc#description hasValue "allow to model the sequence and cardinality of messages in a message exchange"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasDescription ofType (0 1) part

concept *messageExchange*

nonFunctionalProperties

dc#description hasValue "concrete instance of a message exchange pattern"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasMessages ofType (1 1) message

hasMep ofType (0 1) mep

concept *conversation*

nonFunctionalProperties

dc#description hasValue "number of message exchanges among parties"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasMessageExchanges ofType (1 1) messageExchange

concept state

nonFunctionalProperties

dc#description hasValue "condition or situation during the lifetime of a choreography"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

*hasSubStates ofType (0 *) state*

concept task

nonFunctionalProperties

dc#description hasValue "action of sending a message to a party"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasParty ofType (0 1) party

hasMessage ofType (0 1) message

concept action

nonFunctionalProperties

dc#description hasValue "atomic task"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasTask ofType (0 1) task

concept *booleanExpression*

nonFunctionalProperties

dc#description hasValue "boolean expression"

endNonFunctionalProperties

concept *event*

nonFunctionalProperties

dc#description hasValue "occurrence of an stimulus that has a location in time and space"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasBooleanExpression ofType (0 1) booleanExpression

concept *rule*

nonFunctionalProperties

dc#description hasValue "if booleanExpression then state"

endNonFunctionalProperties

concept *guardCondition*

nonFunctionalProperties

dc#description hasValue "defines transitions among states by means of rules"

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasRule ofType (0 1) rule

concept *guardTransition*

nonFunctionalProperties

dc#description hasValue "defines relations among states by means of events, guardConditions and actions"

endNonFunctionalProperties

hasName ofType (1 1) name
hasURI ofType (1 1) uri
*hasEvents ofType (0 *) event*
hasGuardCondition ofType (0 1) guardCondition
*hasActions ofType (1 *) action*

concept part

nonFunctionalProperties

dc:description hasValue "allows establishing the link among a set of guarded transitions and a message exchange"

endNonFunctionalProperties

hasName ofType (1 1) name
hasURI ofType (1 1) uri
hasMessageExchange ofType (0 1) messageExchange
*hasGuardTransitions ofType (1 *) guardTransition*

concept choreography

nonFunctionalProperties

dc:description hasValue "set of parts that govern the message exchange among parties in a conversation"

endNonFunctionalProperties

hasName ofType (1 1) name
hasURI ofType (1 1) uri
hasConversation ofType (0 1) conversation
*hasParts ofType (1 *) part*

concept type

nonFunctionalProperties

dc:description hasValue "type of the logic box "

endNonFunctionalProperties

concept sequenceNumber

nonFunctionalProperties

dc#description hasValue "sequence in which boxes are used in a logic diagram"

endNonFunctionalProperties

concept ontologyMapping

nonFunctionalProperties

dc#description hasValue "mapping among the domain ontology used by the initiating party and the one used by the answering service "

endNonFunctionalProperties

hasURI ofType (1 1) uri

concept type

nonFunctionalProperties

dc#description hasValue "type of logic box "

endNonFunctionalProperties

axiom allowedBoxTypes

definedBy

!- ?x memberOf type and not ?x = refinerBox

and not ?x = mergeBox

and not ?x = splitBox

and not ?x = selectBox

and not ?x = addBox

concept logicBox

nonFunctionalProperties

dc#description hasValue "allows to solve a number of heterogeneities in the messages exchanged by parties "

endNonFunctionalProperties

hasName ofType (1 1) name

hasURI ofType (1 1) uri

hasType ofType (1 1) type

hasSequenceNumber ofType (0 1) sequenceNumber

*hasInputMessages ofType (0 *) message*
hasInputMep ofType (0 1) mep
hasOutputMep ofType (0 1) mep
hasOntologyMapping ofType (0 1) ontologyMapping

concept *logicDiagram*

nonFunctionalProperties

dc#description hasValue "set of interconnected logic boxes that model the relation among the message exchanges used by interacting parties "

endNonFunctionalProperties

hasName ofType (1 1) name
hasURI ofType (1 1) uri
hasInputMessageExchange ofType (0 1) messageExchange
hasOutputMessageExchange ofType (0 1) messageExchange
*hasLogicBoxes ofType (0 *) logicBox*

concept *logicGroup*

nonFunctionalProperties

dc#description hasValue "conceptual entity that allows to put together a number of logic diagrams that model a conversation"

endNonFunctionalProperties

hasName ofType (1 1) name
hasURI ofType (1 1) uri
hasConversation ofType (0 1) conversation
*hasLogicDiagrams ofType (0 *) logicDiagram*