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

Deliverable

WP 5: Service Mediation
D5.5
Business Data and Process-Level Mediation Module
Prototype v2

Emilia Cimpian
Maciej Zaremba
Brahmananda Sapkota
John Domingue
Liliana Cabral

January 16th, 2005
EXECUTIVE SUMMARY

The Business Data and Process-Level Mediation Module Prototype v2 addresses the processes heterogeneity problem, which may occur during the communication between two partners (i.e., the requestor and the provider of a service, two requestors or two providers). The processes considered are the public processes of the two participants in a conversation, expressing their behaviour as Web Service Modeling Ontology (WSMO\(^1\)) choreographies [6]. This version of the prototype exploits the results of the first mediation module prototype [4], by using the services of the data mediation module for transforming the instances sent by one of the partners using its own ontology into instances expressed in terms of the targeted partner ontology.

This deliverable presents details about the Process Mediator components, about the mismatches they can address, and also an example of a pair of heterogeneous choreographies between which mediation can take place.

This deliverable presents the two Process Mediation prototypes developed in DIP, the IRS-III\(^2\) Process Mediator and the WSMX\(^3\) Process Mediator. For fulfilling their functionality, both of these prototypes rely on run-time data mediation components, developed as part of DIP deliverable 5.4 [4]. The WSMX Process Mediator also uses the WSML Reasoner developed in DIP work package I [5].

These prototypes will be used by the DIP use-cases (work packages 8, 9, and 10), and could also prove useful during the service composition. In work package 5, we will investigate this possibility in deliverable 5.8 (Integration of Mediation and Composition in Real World Scenario). We will also address the problem of interoperability between these two prototypes, that is, the way one of them can take advantage of the results returned by the other one.

This document is addressed to any partner interested in the research area of process mediation, service invocation and service composition.

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.

---

1. [www.wsmo.org](http://www.wsmo.org)
# Business Data and Process-Level Mediation Module Prototype v2

## Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>Acronym</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP6 – 507483</td>
<td>DIP</td>
<td></td>
</tr>
</tbody>
</table>

**Full title**: Data, Information, and Process Integration with Semantic Web Services

**Project URL**: [http://dip.semanticweb.org](http://dip.semanticweb.org)

**Document URL**: 

**EU Project officer**: Kai Tullius

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>Business Data and Process-Level Mediation Module Prototype v2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work package Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Service Mediation</td>
</tr>
</tbody>
</table>

**Date of delivery**

<table>
<thead>
<tr>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>M 24</td>
<td>M 24</td>
</tr>
</tbody>
</table>

**Status**

- version. 0.08
- final

**Nature**

- Prototype ✔
- Report □
- Dissemination □
- Ontology □

**Dissemination Level**

- Public □
- Consortium ✔

**Authors (Partner)**

- Emilia Cimpian (NUIG), Maciej Zaremba (NUIG), Brahmananda Sapkota (NUIG), John Domingue (OU), Liliana Cabral (OU)

**Responsible Author**

- Emilia Cimpian
- Partner: NUIG
- Email: emilia.cimpian@deri.org
- Phone: +353-91-495113

**Abstract (for dissemination)**

One of the biggest obstacles Web Services have to overcome in the attempt to exploit the true potential of the World Wide Web is heterogeneity. Caused by the nature of the Web itself, heterogeneity problems occur both at data level and at behavioural level of business logics, message exchange protocol and Web Service invocation.

Process mediation is one of the crucial points on the road towards establishing new, ad-hoc cooperation on the web between various business partners. If semantic enhanced data enabled dynamic solutions for coping with data heterogeneity, semantically enhanced Web Services can do the same for behavioural heterogeneity. Based on the Web Service Modeling Ontology (WSMO) specifications that offer support in semantically describing Web Services, we propose a solution that acts on these semantic descriptions and offers the means for defining what we call a Process Mediator. Such a mediator acts on the public processes (represented as WSMO choreographies) of the parties involved in a communication and adjust the bi-directional flow of messages to suit the requested/expected behaviour of each party.

**Keywords**

- Process, Choreography, Behaviour, Mediation

**Version Log**

ii 1/16/2006
<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.11.2005</td>
<td>001</td>
<td>Emilia Cimpian</td>
<td>First version</td>
</tr>
<tr>
<td>01.12.2005</td>
<td>002</td>
<td>Emilia Cimpian</td>
<td>Initial content of the deliverable</td>
</tr>
<tr>
<td>07.12.2005</td>
<td>003</td>
<td>Brahmanand Sapkota</td>
<td>Contribution on Section 3</td>
</tr>
<tr>
<td>08.12.2005</td>
<td>004</td>
<td>Emilia Cimpian</td>
<td>OU part included (Section 2) -&gt; content was provided by Liliana Cabral</td>
</tr>
<tr>
<td>08.12.2005</td>
<td>005</td>
<td>Maciej Zaremba</td>
<td>Contribution to Section 3 and Annexes</td>
</tr>
<tr>
<td>08.12.2005</td>
<td>006</td>
<td>Emilia Cimpian</td>
<td>Final changes before the review</td>
</tr>
<tr>
<td>19.12.2005</td>
<td>007</td>
<td>Emilia Cimpian</td>
<td>Update template and changes required by reviewers</td>
</tr>
<tr>
<td>11.01.2006</td>
<td>008</td>
<td>Liliana Cabral</td>
<td>Updated changes required by reviewers</td>
</tr>
<tr>
<td>16.01.2006</td>
<td>009</td>
<td>Emilia Cimpian</td>
<td>Minor editorial change required by final review</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reviewer</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laurent Henocque</td>
<td><a href="mailto:laurent.henocque@gmail.com">laurent.henocque@gmail.com</a></td>
<td></td>
</tr>
<tr>
<td>Partner ILOG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mick Kerrigan</td>
<td><a href="mailto:mick.kerrigan@deri.org">mick.kerrigan@deri.org</a></td>
<td>+353 91 495285</td>
</tr>
<tr>
<td>Partner NUIG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>National University of Ireland Galway</td>
<td>NUIG</td>
<td>Dr. Sigurd Harand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Digital Enterprise Research Institute (DERI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>National University of Ireland, Galway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Galway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ireland</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:sigurd.harand@deri.org">sigurd.harand@deri.org</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +353 91 495112</td>
</tr>
<tr>
<td>Fundacion De La Innovacion.Bankinter</td>
<td>Bankinter</td>
<td>Monica Martinez Montes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fundacion de la Innovacion. Bankinter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Paseo Castellana, 29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28046 Madrid, Spain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:mmtnez@bankinter.es">mmtnez@bankinter.es</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: 916234238</td>
</tr>
<tr>
<td>Berlecon Research GmbH</td>
<td>Berlecon</td>
<td>Dr. Thorsten Wichmann</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Berlecon Research GmbH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oranienburger Str. 32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10117 Berlin, Germany</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:tw@berlecon.de">tw@berlecon.de</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +49 30 2852960</td>
</tr>
<tr>
<td>British Telecommunications Plc.</td>
<td>BT</td>
<td>Dr John Davies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BT Exact (Orion Floor 5 pp12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adastral Park Martlesham</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ipswich IP5 3RE, United Kingdom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:john.nj.davies@bt.com">john.nj.davies@bt.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +44 1473 609583</td>
</tr>
<tr>
<td>Swiss Federal Institute of Technology, Lausanne</td>
<td>EPFL</td>
<td>Prof. Karl Aberer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distributed Information Systems Laboratory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>École Polytechnique Fédérale de Lausanne</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bât. PSE-A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1015 Lausanne, Switzerland</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:Karl.Aberer@epfl.ch">Karl.Aberer@epfl.ch</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +41 21 693 4679</td>
</tr>
<tr>
<td>Essex County Council</td>
<td>Essex</td>
<td>Mary Rowlatt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Essex County Council</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PO Box 11, County Hall, Duke Street</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chelmsford, Essex, CM1 1LX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>United Kingdom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:mary@essexcc.gov.uk">mary@essexcc.gov.uk</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +44 (0)1245 436524</td>
</tr>
<tr>
<td>Forschungszentrum Informatik</td>
<td>FZI</td>
<td>Andreas Abecker</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Forschungszentrum Informatik</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Haid-und-Neu Strasse 10-14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>76131 Karlsruhe, Germany</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:abecker@fzi.de">abecker@fzi.de</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +49 721 9654 0</td>
</tr>
<tr>
<td>Partner</td>
<td>Acronym</td>
<td>Contact</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Institut für Informatik, Leopold-Franzens</td>
<td>UIBK</td>
<td>Prof. Dieter Fensel&lt;br&gt;Institute of computer science&lt;br&gt;University of Innsbruck&lt;br&gt;Technikerstr. 25&lt;br&gt;A-6020 Innsbruck, Austria&lt;br&gt;Email: <a href="mailto:dieter.fensel@deri.org">dieter.fensel@deri.org</a>&lt;br&gt;Tel: +43 512 5076485</td>
</tr>
<tr>
<td>Universität Innsbruck</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILOG SA</td>
<td>ILOG</td>
<td>Christian de Sainte Marie&lt;br&gt;9 Rue de Verdun, 94253&lt;br&gt;Gentilly, France&lt;br&gt;Email: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a>&lt;br&gt;Tel: +33 1 49082981</td>
</tr>
<tr>
<td>inubit AG</td>
<td>Inubit</td>
<td>Torsten Schmale&lt;br&gt;inubit AG&lt;br&gt;Lützowstraße 105-106&lt;br&gt;D-10785 Berlin&lt;br&gt;Germany&lt;br&gt;Email: <a href="mailto:ts@inubit.com">ts@inubit.com</a>&lt;br&gt;Tel: +49 30726112 0</td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>iSOCO</td>
<td>Dr. V. Richard Benjamins, Director R&amp;D&lt;br&gt;Intelligent Software Components, S.A.&lt;br&gt;Pedro de Valdivia 10&lt;br&gt;28006 Madrid, Spain&lt;br&gt;Email: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a>&lt;br&gt;Tel. +34 913 349 797</td>
</tr>
<tr>
<td>NIWA WEB Solutions</td>
<td>NIWA</td>
<td>Alexander Wahler&lt;br&gt;NIWA WEB Solutions&lt;br&gt;Niederacher &amp; Wahler OEG&lt;br&gt;Kirchengasse 13/1a&lt;br&gt;A-1070 Wien&lt;br&gt;Email: <a href="mailto:wahler@niwa.at">wahler@niwa.at</a>&lt;br&gt;Tel:+43(0)1 3195843-11</td>
</tr>
<tr>
<td>The Open University</td>
<td>OU</td>
<td>Dr. John Domingue&lt;br&gt;Knowledge Media Institute&lt;br&gt;The Open University, Walton Hall&lt;br&gt;Milton Keynes, MK7 6AA&lt;br&gt;United Kingdom&lt;br&gt;Email: <a href="mailto:l.b.domingue@open.ac.uk">l.b.domingue@open.ac.uk</a>&lt;br&gt;Tel.: +44 1908 655014</td>
</tr>
<tr>
<td>The Open University</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAP AG</td>
<td>SAP</td>
<td>Dr. Elmar Dorner&lt;br&gt;SAP Research, CEC Karlsruhe&lt;br&gt;SAP AG&lt;br&gt;Vincenz-Priessnitz-Str. 1&lt;br&gt;76131 Karlsruhe, Germany&lt;br&gt;Email: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a>&lt;br&gt;Tel: +49 721 6902 31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td>Contact Information</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Sirma Al Ltd.</td>
<td>Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD&lt;br&gt;Office Express IT Centre, 3rd Floor&lt;br&gt;135 Tzarigradsko Chausse&lt;br&gt;Sofia 1784, Bulgaria&lt;br&gt;Email: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a>&lt;br&gt;Tel.: +359 2 9768 303</td>
<td></td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg&lt;br&gt;Unicorn Solutions Ltd, Malcha Technology Park 1&lt;br&gt;Jerusalem 96951&lt;br&gt;Israel&lt;br&gt;Email: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a>&lt;br&gt;Tel.: +972 2 6491111</td>
<td></td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Pieter De Leenheer&lt;br&gt;Starlab- VUB&lt;br&gt;Vrije Universiteit Brussel&lt;br&gt;Pleinlaan 2, G-10&lt;br&gt;1050 Brussel ,Belgium&lt;br&gt;Email: <a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a>&lt;br&gt;Tel.: +32 (0) 2 629 3749</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF KEY WORDS/ABBREVIATIONS

Process
Choreography
Behaviour
Mediation
WSMO – Web Service Modeling Ontology
WSML – Web Service Modeling Language
WSMX – Web Service Execution Environment
IRS – Internet Reasoning Service
# TABLE OF CONTENTS

**EXECUTIVE SUMMARY** ............................................................................................................. 1  
**LIST OF KEY WORDS/ABBREVIATIONS** .............................................................................. VII  
**TABLE OF CONTENTS** ........................................................................................................ VI  
1 INTRODUCTION .......................................................................................................................... 1  
2 IRS-III MEDIATION FRAMEWORK AND PROCESS MEDIATION ................................... 2  
  2.1 Process Mediation .................................................................................................................. 3  
  2.2 Installation and Usage ............................................................................................................. 4  
  2.3 Requirements .......................................................................................................................... 4  
3 WSMX PROCESS MEDIATOR ................................................................................................. 5  
  3.1 Overview of related prototypes and DIP deliverables .............................................................. 5  
  3.2 Architecture ........................................................................................................................... 8  
    3.2.1 Interaction with the environment ...................................................................................... 9  
    3.2.2 Interaction between components .................................................................................... 10  
  3.3 Sub-components ..................................................................................................................... 10  
    3.3.1 Core Component .............................................................................................................. 10  
    3.3.2 Validator ......................................................................................................................... 11  
    3.3.3 Internal Repository .......................................................................................................... 11  
    3.3.4 Reasoner Wrapper ............................................................................................................ 12  
  3.4 Requirements .......................................................................................................................... 12  
4 MAIN DIFFERENCES BETWEEN IRS-III AND WSMX PROCESS MEDIATORS .......... 14  
5 CONCLUSIONS AND FURTHER DEVELOPMENTS ............................................................ 14  
-REFERENCES ................................................................................................................................. 14  
ANNEX 1. VIRTUAL TRAVEL AGENCY EXAMPLE .................................................................... 15  
  Travel Requestor Ontology, Goal and Generated Instances ...................................................... 15  
  Travel Provider Ontology, Service Description and Generated Instances ............................. 22  
  Mapping Rules used by the Run-Time Data Mediator ............................................................. 28  
  Log4J Logs .................................................................................................................................. 33  
ANNEX 2. WSMX PROCESS MEDIATOR JAVA DOC .............................................................. 37  
  Interface ProcessMediator .......................................................................................................... 37  
  Interface Reasoner ...................................................................................................................... 39  
  Interface Validator ....................................................................................................................... 40  
  Interface DataStorage .................................................................................................................. 40
ANNEX 3. IRS-III PROCESS MEDIATOR EXAMPLE ................................................................. 42

LIST OF FIGURES
Figure 1: Mediation Framework for IRS III ...................................................................... 2
Figure 2: Mediation between Two Sub-Goals; the Input of Goal1 is Transformed in One Input of Goal2 ........................................................................................................ 4
Figure 3: Overview of WSMX Data Mediation ................................................................. 6
Figure 4: Resolvable Message Mismatches ...................................................................... 7
Figure 5: WSMX Process Mediator Architecture ............................................................. 9

LIST OF TABLES
Table 1: Requestor’s Ontology .......................................................................................... 15
Table 2: Travel Request Goal ........................................................................................... 17
Table 3: Instances Sent by the Requestor ......................................................................... 21
Table 4: Travel Provider Ontology .................................................................................. 22
Table 5: Travel Provider Service ...................................................................................... 24
Table 6: Instances Sent by the Provider .......................................................................... 27
Table 7: Mapping Rules for the Requestor to Provider Communication ....................... 28
Table 8: Mapping Rules for the Provider to Requestor Communication ....................... 31
Table 9: Process Mediator log .......................................................................................... 33
Table 10: Requester’s Thread log ..................................................................................... 35
Table 11: Provider’s Thread log ....................................................................................... 36
1 INTRODUCTION

In a dynamic environment such as the World Wide Web\(^4\), which is being used more and more for application to application communication, solutions for allowing dynamic interoperability have become a must. This is because the service providers and the service requestors usually use different formalisms for representing their data, as well as different communication patterns in order to communicate with each other.

In this context, one of the main focuses of the DIP community is providing means for both data mediation and process (behavioural) mediation. The development of the Business Data and Process-Level Mediation Module Prototype v2 takes advantage of the work carried out on the development of the first Business Data and Process-Level Mediation Module Prototype [4].

This deliverable consists of two main parts: Section 2 presents details about the IRS-III Mediation Framework and the way it addresses process mediation, and Section 3 presents details about the WSMX Process Mediator.

In IRS-III, process mediation takes place during the invocation and composition of Semantic Web Services. IRS-III process mediator expects the requester to follow the choreography of the provider during invocation. For composition, the requester can publish WSMO mediator descriptions with declared mediation services, which can provide dataflow within the orchestration as well as allow transforming inputs between sub-goals into the right format/value.

In WSMX, the mediation of behaviours is applied when the communication patterns of the two participants are different, in which case one of them has to adjust to the other’s communication pattern (i.e., it has to change its process execution in order to match the other party’s specifications). The adjustment of the different patterns in order to make them match is called process mediation, and in WSMX this adjustment happens in neither of the involved parties’ choreographies, but is handled by the Process Mediator [3]. That is, each time a message is sent by one of the two parties involved in the conversation, the Process Mediator has to determine if the message is expected by the other party. The mediator also has to consider situations when only part of a message or a combination of this message with a previously received one is expected, which is done by analyzing the choreographies of the parties.

This document also contains three Annexes. The first one presents an example of a pair of heterogeneous choreographies, containing all the types of mismatches addressable by the WSMX process mediator. The second annex contains the Java doc for the WSMX process mediator. The third annex contains an illustration scenario on the use of WSMO mediator descriptions by IRS-III.

---

\(^4\) http://www.w3.org/
2 IRS-III MEDIATION FRAMEWORK AND PROCESS MEDIATION

The IRS-III mediation framework implements data mediation, goal mediation and process mediation of Semantic Web Services. The main objective is to provide mediation components which solve a number of different types of mismatches by reasoning over given Goal, Web Service and Mediator descriptions.

In particular, the following sub-section will explain in more details the use of WSMO conceptual models by the process mediator component.

![Figure 1: Mediation Framework for IRS III](image)

Figure 1 illustrates the main architecture components incorporated in the mediation framework of IRS-III. In the steps below we describe the overall sequence of mediation activities taking place during selection, composition and invocation of Semantic Web Services.

1. The Process Mediator establishes an interaction with a deployed web service (code) by executing its Web Service choreography through the Choreography Interpreter. The Process Mediator performs the lifting and lowering of data provided by the Web Service grounding and is able to create the communication messages corresponding to the choreography communication primitives. It keeps the state of the communication throughout operation calls executed via the Invoker component.

2. The Process Mediator component also executes the orchestration of a composite Web Service using the Orchestration Interpreter. It keeps the state of the orchestration (control and data flow) between invocations of sub-Goals. The Process Mediator searches for GG-mediators connecting sub-Goals in the orchestration. The types of mismatches that can occur are: a) output types of a sub-goal are different from the input types of the target sub-Goal; b) output values of a sub-goal are in a different order from the inputs of the target sub-Goal; c) the output of a sub-Goal has to be split or concatenated into the inputs of the target sub-goals.

3. The Data Mediator component is used by the Goal Mediator and by the Process Mediator for mapping data across domain ontologies. It executes the mapping rules of OO-mediators used by other WSMO elements. Details of the IRS-III Data Mediator can be found on deliverable D5.4 [4].

4. The Goal Mediator searches for WG-Mediators whose source component matches the current Goal when IRS-III receives an achieve-goal request from a client application. It selects the first targeted Web-Service which matches the
requested capabilities (input types, preconditions, assumptions, non-functional properties etc). The types of mismatches that can occur are: a) the input types of a Goal are different from the input types of the target Web Service; and b) Web Services have more inputs than the Goal.

2.1 Process Mediation

The Process mediation component of IRS-III handles mismatches that occur during the invocation or composition of a Web Service. IRS-III either executes the choreography (interaction protocol) of a single Web Service or the orchestration of a composed Web Service, using the values provided by the Goal inputs. Moreover, the Process mediator has to execute the choreography of each single Web Service in the Orchestration.

In IRS-III the choreography of a Web Service describes how to interact with a deployed service (client choreography). A set of rules (guarded transitions) in the choreography are used to specify the flow of operations required for realizing the specific functionality of the Web Service. The Process Mediator uses the Web Service grounding for creating the communication messages based on the operations declared at the conceptual level.

A choreography is provided to interact with a single Web Service. By interpreting the choreography and grounding, the Process mediator component can send messages to the service in the right order and format on behalf of the client. When a Web Service is a composite of multiple services an orchestration has to be provided instead. Nevertheless, its input values have to be passed to the orchestration and the result of the orchestration has to be passed back to the Web Service. The orchestration follows the decomposition of Goals into sub-Goals and uses GG-mediators for connecting sub-goals and mediating the order and types of inputs between them. A GG-mediator is a WSMO semantic description (like a WSMO Goal or Web Service) connecting a source WSMO component (Goal) to a target WSMO component (Goal), therefore there may be many GG-mediators in an orchestration description. More details about IRS-III choreography and orchestration languages are described in deliverables D3.4 and D3.5.

We illustrate in the following the role of a GG-mediator during orchestration (see Figure 2). The provider of a Web service describes the orchestration through control-flow mechanisms, for instance: (sequence G1 G2 M1). The Sequence control command executes the given sub-goals (G1 and G2) in sequence. Figure 2 shows the graphical representation of the GG-mediator connecting G1 to G2. This mediator supports the data flow between the sub-goals and the necessary transformations. The source goal (G1) produces one output (E1), which is transformed by the mediation service in one input (E2) used by the target Goal (G2). During the execution of the orchestration the input values (SC, TC, A) received by the current invoked Goal are sent onto the sub-goals through matching, then the associated GG-mediator (M1) is used to connect and forward results between sub-goals providing the necessary transformations through the mediation service.
Figure 2: Mediation between Two Sub-Goals; the Input of Goal1 is Transformed in One Input of Goal2

WW-mediators can be used in a similar way to GG-mediators by the Process Mediator. In this case, the WW-mediator can provide mappings between the input values of the current Web Service and the Web Services in the orchestration.

2.2 Installation and Usage

The mediation component described above is part of the IRS-III architecture, which is composed of a server, a client and a publisher platform. The IRS-III server is written in Lisp and available as an executable file. The Publisher is a Web application and the client is a Java API. A Browser (GUI), which uses the Java API, has also been developed for users to navigate the library of models (ontologies) and to edit WSMO descriptions. The browser can also be used for publishing and invoking individual services. Currently, there is no tool for automatically generating the mapping rules of a mediator, therefore these have to be written manually in OCML using the IRS-III browser.

A distribution package for IRS-III with the components above, including a user guide and test applications is freely available from the website: http://kmi.open.ac.uk/projects/irs. This package is not for commercial purposes and no source code is available.

Detailed installation instructions are part of the distribution package.

2.3 Requirements

**Nature:** A knowledge-based execution environment for WSMO, including Mediators.

**Interfaces:** A Java API and a browser/editor (GUI).

**Client Platform:** JDK 1.4.2 or 1.5.

**Required Libraries:** Apache TomCat (or another Web Server), SOAP, and Ant 1.5.* (http://apache.org/)
3 WSMX PROCESS MEDIATOR

This chapter provides details about the WSMX Process Mediator. The relation between this prototype and the previous efforts in DIP is covered in sub-section 3.1.

3.1 Overview of related prototypes and DIP deliverables

The development of the WSMX Process Mediator prototype is based on DIP deliverable 5.3a – Business Process-Level Mediation Module Specification [3], and also takes advantage of the work carried on for the development of the first Business Data and Process-Level Mediation Module Prototype [4]. This section explains the relation between this prototype and the previously mentioned deliverables.

D5.4 - Business data and process-level mediation module prototype v1

The focus of the first Business Data and Process-Level Mediation Module Prototype was the development of a data mediation tool, capable to compensate for the mismatches between two different ontologies. For this purpose two different ontology to ontology prototypes were developed, one of them being integrated in the Web Service Execution Environment (WSMX), and the other one in the Internet Reasoning Service (IRS).

Considering that two participants in a conversation may not only have different communication patterns, but also may use different ontologies, the Process Mediator needs the services of an external Data Mediator in order to provide its functionality. Figure 3 presents an overview of the WSMX Data Mediator, which consists of a design time tool (offering support to a domain expert for creating the mappings), and a run-time tool (the Execution Environment), able to execute the previously created mappings.

The Data Mediator run-time component is invoked by the Process Mediator by providing it with the instance(s) to be mediated and with the identifiers of the two ontologies (alternatively the entire ontologies could be sent as parameters). The returned result will be the mediated instance(s) in terms of the targeted partner ontology.
D5.3a - Business Process-level Mediation Module Specification

The Business Process-level Mediation Module Specification proposes a solution for coping with the differences in the way a requester wants to consume the functionality of a Web service and the way this functionality is made available by the Web service to the requester. The description of the expected/requested behaviour of the two parties (which is in fact a formalization of their public business processes) is made using WSMO choreographies. Using these descriptions and the services of a data mediator (to solve data heterogeneity problems) the authors introduce the Process Mediator, a system able to adjust the two parties' behaviour, enabling their communication.

The authors identify five types of behavioural mismatches that a Process Mediator should address (Figure 4):

a) **Stopping an unexpected message** (Figure 4: a) – If one of the partners sends a message that the other one does not want to receive, the mediator should just dispose of this message.

b) **Inverting the order of messages** (Figure 4: b) – If one of the partners sends the messages in a different order than the other partner expects, the messages that are not yet expected will be stored and sent when needed. This case involves the previous one, the first message being actually stored when received, and not immediately sent to the targeted partner.

c) **Splitting a message** (Figure 4: c) – If one of the partners sends in a single message pieces of information that the other one expects to receive in different messages, the information can be split and sent in a sequence of separate messages. The splitting of a certain instance into several different instances, that

---

5 From [4]
may appear during the data mediation process (in case an instance expressed in terms of the sender’s ontology is mapped into multiple instances in terms of the targeted ontology), is only a subset of the message splitting, since a message can consist of multiple instances.

d) **Combining messages** (Figure 4: d) – If one of the partners expects a single message, containing information sent by the other one in multiple messages, the information can be combined into a single message. This combined message may also contain data that was already transmitted. As with the previously described case, combining messages may also involve, but it is not limited to, combining several instances in a single one during the data mediation process (the result of mapping a number of instances from the sender’s ontology could be a single instance in terms of the targeted partner ontology).

e) **Sending a dummy acknowledgement** (Figure 4: e) – If one of the partners expects an acknowledgement for a certain message, and the other partner does not intend to send it, even if it receives the message, an acknowledgment can be automatically generated and sent to the partner which requires it.

Out of the five types of mismatches, the WSMX Process Mediation prototype is able to address only the first four (Stopping an unexpected message, Inverting the order of messages, Splitting a message and Combining messages). The last one (Sending a dummy acknowledgement) cannot yet be solved, but the work on this prototype will continue as part of DIP deliverable 5.8 (Integration of Mediation and Composition in

---

6 From [3]
Real World Scenario), and we intend to address this problem as well. A complete list of the unresolvable message mismatches is provided in [3]

An example of a pair of choreographies illustrating these mismatches is provided in Annex 1. Virtual Travel Agency Example

The architecture of this prototype also follows the specification from D5.3a. However, some small adjustments were required:

− The Choreography Parser sub-component was renamed Validator, since it does not have to parse the two choreographies, but only to determine if a certain instance is expected or not by the targeted partner (valid or not from the conversation point of view);

− The Internal Repository sub-components remains as specified in D5.3a, the only difference being that it has to maintain two lists, one for each of the participants in the conversation;

− The WSML Reasoner sub-component is an important part of the architecture, but we did not implement a new reasoner since there is already an existing one in the DIP architecture [5]. Instead, we created a wrapper that will provide the reasoner with the appropriate inputs, and will interpret the results.

− A new sub-component was introduced: Core Component for the process mediator. Its task is to manage the interaction with other DIP components, and to coordinate the activities of the Process Mediator internal sub-components.

3.2 Architecture

The Process Mediator architecture follows the component based design principle. It consists of four main components each having different functional characteristics. These are: Core Component, Reasoner Wrapper, Validator and Internal Repository. The interaction between the Process Mediator and external components takes place through the Core Component. The functional behaviour of these components is explained later in this document. These components collectively form the Process Mediator architecture as shown in Figure 5. The figure also shows the relationship between Process Mediator and other WSMX components.

In the following sub-sections the Process Mediator interactions with the environment as well as the internal interactions between the Process Mediator’s sub-components are described.
3.2.1 Interaction with the environment

The environment of the Process Mediator is defined as a set of external components that interact with the Process Mediator either to make use of the offered service or to offer services required by the Process Mediator. The Choreography Engine, the Data Mediator, and the Reasoner form the Process Mediator’s environment.

The Choreography Engine interacts with the Process Mediator in order to make use of the offered service (i.e., process mediation). It sends a message together with the choreography instances (choreography copies) of the sender and of the targeted partner to the Process Mediator. The result of the process mediation is one or more messages expected by one of the partners, together with the choreography instance of the partner that expects that message(s)\(^7\).

The Process Mediator interacts with the Data Mediator in order to request data mediation service. If the ontologies used to define the messages are not homogeneous, the Process Mediator invokes the Data Mediator to solve these heterogeneity problems.

\(^7\) Both the Choreography Engine API and the WSMO Choreography API are available but there is no generic Choreography Engine implementation that would be able to execute the choreography specifications of the interacting parties. This situation required hardwiring the choreographies of interacting parties (i.e., the Requestor and the Provider).

Therefore, instead of generic choreographies of interacting parties, predefined Java Threads representing their choreographies are provided. Choreographies are created within them and the threads are aware of their specifics and execute them according to their predefined internal logic. If the generic Choreography Engine existed at this time, they would be executed according to their logic determined during the runtime, without making any a priori assumptions with respect to their execution logic.
It sends the instances contained in the message, expressed in terms of one partner’s ontology, and expects the mediated instances, expressed in terms of the targeted partner’s ontology.

The interaction between the Process Mediator and the Reasoner takes place for using the reasoning service required by the Process Mediator.

### 3.2.2 Interaction between components

The overall functionality of the Process Mediator i.e., the process mediation service is achieved by the interaction between the components. In the architecture given in Figure 1, the Core Component interacts with:

- the Validator component, in order to validate the instances contained by the incoming message (i.e., to check if they are expected by the targeted partner);
- the Repository component, for storing/retrieving valid instances;
- the Reasoner Wrapper, for invoking the external Reasoner component.

### 3.3 Sub-components

The Process Mediator prototype consists of the following sub-components:

- Core Component,
- Validator,
- Internal Repository,
- Reasoner.

In the following sub-sections, each of these sub-components will be explained in more detail.

#### 3.3.1 Core Component

The Core Component is the central sub-component of the Process Mediator; its task is to manage the interaction with other DIP components (Data Mediator and Choreography Engine), as well as to coordinate the activities of the Process Mediator sub-components.

After being invoked by the Choreography Engine, which provides the message sent by one of the partners and the two choreographies involved in the conversation, the Core Component has to perform the following steps:

1. Invoke the Data Mediator, with the parameters: instances to be mediated, source choreography, target choreography; the result returned are the mediated instance(s) in terms of the targeted partner ontology;

2. Send the mediated instances to the Validator, to check which of them are expected by the targeted partner; the result returned represents a subset of the invocation parameter, containing only the expected instances (valid instances);
3. Store all the valid instances in the Internal Repository;
4. Retrieve all the instances from the Internal Repository corresponding to one of the partners;
5. Invoke the Reasoner (using the Reasoner Wrapper) by providing it with the available instances and the corresponding choreography; the Reasoner will determine which instances are expected at this particular point of the communication;
6. Send the expected instances to the Choreography Engine\(^8\).

### 3.3.2 Validator

The Validator component is used for determining if a certain instance is expected by the targeted partner, at any point of the communication. This information is stored in the state signature of the corresponding choreography: all the concepts that can be instantiated by the communicating partner are marked as in or as shared for the Web Services, and as out or shared for goals, any goal describing the requested choreography, which is the mirror of its own communication pattern [6] (that is, the goal describe the choreography of the Web service able to fulfil the request). For example, when the goal specifies that a certain instance should be sent to the service, the corresponding concept will be marked as in, since the service expects it, and not as out (see Annex 1 for example of such a list).

For performing this checking, the Validator has to:

1. Lift the instance to the level of concepts (i.e., determine the concept that was instantiated);
2. Determine if the concept is part of the in or shared list, for Web Services, or part of the out or shared lists for goals.

### 3.3.3 Internal Repository

The Internal Repository component provides support for in-memory storage of received instances. The instances that the process mediator receives can be sent by the requestor to the provider of the service, or the other way around. These instances are stored separately in two in-memory repositories, namely localWSReceived and localGoalReceived.

The Internal repository supports the following operations:

1. push - the required parameters for the execution of the push operation are the instances that need to be stored and the invocation direction. The invocation direction indicates whether the instance was provided by the Web service or by the goal.

---

\(^8\) Steps 4, 5 and 6 will be performed twice – once for each of the communication partners.
2. remove - the remove operation takes a list of instances and the invocation direction as input parameters and removes those instances from the repository indicated by the invocation direction.

3. get - taking a concept and the invocation direction as input parameters, the get operation returns the set of instances of that concept from the repository indicated by the invocation direction.

4. getAll – the getAll operation takes invocation direction as the only input parameter and returns all the instances from the repository indicated by the invocation direction.

3.3.4 Reasoner Wrapper

The WSML Reasoner utilized by the Process Mediation supports the WSML-Flight variant [2] and has been developed within DIP work package I [5]. WSML-Flight extends WSML-Core in direction of the logic programming, enabling modeling primitives for attributes like value or integrity constraints. What is relevant from the point of view of the applicability of this reasoning in the DIP execution environment is its decidability.

WSML Reasoner supports two different representations, namely description logic and logic programming. As the underlying inference engine it uses the KAON2 implementation, that is, a hybrid reasoner supporting both paradigms. WSMO4J objects are used to interact with the WSML Reasoner, whilst internally they are mapped to KAON2 native representation and any inference is performed on this native representation. However, these mappings to the native KAON2 representation are executed internally and are not exposed in WSML Reasoner API. Execution of these mappings takes a certain amount of time, but it abstracts the developer from the details of the reasoning and provides a seamless WSML-driven approach to both reasoning and querying knowledge base expressed as an ontology.

The Reasoner currently fully supports the if-then choreography constructs, whilst not supporting more complex scenarios involving parallelism and iteration.

The Reasoner Wrapper we developed provides the Reasoner with the appropriate inputs, and interprets the results.

3.4 Requirements

The Process Mediation prototype has been written in Java 1.5. In order to work with the system the latest release of Java is required. It can be downloaded from http://java.sun.com/. Process Mediation is a part of the Web Services Execution Environment (WSMX) and can be checked out from the CVS at Sourceforge (http://sourceforge.net/projects/wsmx/). All required libraries to run the Process Mediation can be also checked out from that CVS repository, thus it is not necessary to download them separately. These libraries are:

- wsmo4j – an object-model for WSML;
- interprolog – intermediary layer allowing to communicate with Flora-2;
- kaon2 – native prolog reasoning engine;
- wsmo4j-wrapper – wrapper for the kaon2 Reasoner allowing the use of wsmo4j objects;
- log4j – provides convenient logging facilities.
4 MAIN DIFFERENCES BETWEEN IRS-III AND WSMX PROCESS MEDIATORS

One main difference between IRS-III and WSMX process mediator is that IRS-III uses WSMO Mediator descriptions (e.g. oo-mediator, gg-mediator) during process mediation, interprets the choreography provided by the Web Service and handles mismatches during orchestration. IRS-III follows goal decomposition within the orchestration and uses gg-mediators for supporting mediation between sub-goals. IRS-III also implements mediation services as WSMO Goals for transforming values between sub-goals. On the other hand, the WSMX Process Mediator uses only the services of the run-time Data Mediator component, assuming that the mappings between different data are already represented in an independent language [4] and only need to be executed. It analyzes two choreography instances (belonging to two participants in a conversation) and resolves the mismatches that may appear during the communication between the two partners.

5 CONCLUSIONS AND FURTHER DEVELOPMENTS

This document presents details about the Business Data and Process-Level Mediation Module Prototype v2 – the Process Mediator component. We illustrated the relation between the Process Mediator and the Data Mediator prototype previously developed. There are two approaches presented in this deliverable, addressing different aspects of the same problem, as illustrated in the previous section.

We intend to improve the functionalities offered by the Process Mediator as part of DIP deliverable 5.8 (Integration of Mediation and Composition in Real World Scenario), when we will also test our results on real use-cases, which will be provided by the work packages 8, 9 and 10.

REFERENCES

ANNEX 1. VIRTUAL TRAVEL AGENCY EXAMPLE

This annex presents a pair of choreographies — the choreography of a Semantic Web Service that offers facilities for trip tickets booking, and the one of a requestor of such a service. The communication between the goal and the Web service which have these choreographies is hampered by a certain number of mismatches that can be solved by the Process Mediator prototype. The annex also contains the ontologies used by the service provider and requestor, as well as the mapping rules used by the Data Mediator for mediating the exchanged instances.

Travel Requestor Ontology, Goal and Generated Instances

Table 1: Requestor’s Ontology

<table>
<thead>
<tr>
<th>Travel Request Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wsmlVariant</strong></td>
</tr>
<tr>
<td><strong>namespace</strong></td>
</tr>
<tr>
<td><strong>ontology</strong></td>
</tr>
<tr>
<td><strong>nonFunctionalProperties</strong></td>
</tr>
</tbody>
</table>
  _"http://purl.org/dc/elements/1.1#title" hasValue "Travel Request Ontology"
  _"http://purl.org/dc/elements/1.1#language" hasValue "English"
  _"http://purl.org/dc/elements/1.1#creator" hasValue "Emilia Cimpian"

| **concept** | ticket |
|--------------------------|
| **nonFunctionalProperties** |  
  _"http://purl.org/dc/elements/1.1#description" hasValue "Ticket concept"
| **endNonFunctionalProperties** |
| **departure_Time** | impliesType time |
| **departure_Date** | impliesType date |

---

9 The choreographies, as well as the underlying ontologies are described using Web Service Modeling Language [http://www.wsmo.org/wsml](http://www.wsmo.org/wsml); DIP Work Package 3 is also respecting this syntax.
<table>
<thead>
<tr>
<th>Concept</th>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>price</strong></td>
<td>nonFunctionalProperties</td>
<td>&quot;<a href="http://purl.org/dc/elements/1.1#description">http://purl.org/dc/elements/1.1#description</a>&quot; hasValue &quot;Price concept&quot;</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td>nonFunctionalProperties</td>
<td>&quot;<a href="http://purl.org/dc/elements/1.1#description">http://purl.org/dc/elements/1.1#description</a>&quot; hasValue &quot;Date concept&quot;</td>
</tr>
<tr>
<td><strong>time</strong></td>
<td>nonFunctionalProperties</td>
<td>&quot;<a href="http://purl.org/dc/elements/1.1#description">http://purl.org/dc/elements/1.1#description</a>&quot; hasValue &quot;Time concept&quot;</td>
</tr>
<tr>
<td><strong>person</strong></td>
<td>nonFunctionalProperties</td>
<td>&quot;<a href="http://purl.org/dc/elements/1.1#description">http://purl.org/dc/elements/1.1#description</a>&quot; hasValue &quot;Person concept&quot;</td>
</tr>
<tr>
<td><strong>creditCard</strong></td>
<td>nonFunctionalProperties</td>
<td>&quot;<a href="http://purl.org/dc/elements/1.1#description">http://purl.org/dc/elements/1.1#description</a>&quot; hasValue &quot;Credit card concept&quot;</td>
</tr>
</tbody>
</table>

- **ticket_Price** impliesType _integer
- **year** impliesType _integer
- **month** impliesType _integer
- **day** impliesType _integer
- **hour** impliesType _integer
- **minutes** impliesType _integer
- **name** impliesType _string
- **cc_Owner** impliesType person
- **cc_Expiration_Date** impliesType date
- **cc_Number** impliesType _integer
concept station

nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#description" hasValue "Station concept\n"
endNonFunctionalProperties

start_Location impliesType _boolean
destination_Location impliesType _boolean
name impliesType _string

Table 2: Travel Request Goal

<table>
<thead>
<tr>
<th>Travel Request Goal</th>
</tr>
</thead>
</table>

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace { _"http://www.examples.org/Goals/TravelRequest#",
    tro _"http://www.examples.org/ontologies/TravelRequestOntology"}

webService _"http://www.examples.org/goals/travelRequest"

nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#title" hasValue "Travel Request\n"
    _"http://purl.org/dc/elements/1.1#language" hasValue "English\n"
    _"http://purl.org/dc/elements/1.1#creator" hasValue "Emilia Cimpian\n"
endNonFunctionalProperties

importsOntology {
    _"http://www.example.org/ontologies/TravelRequestOntology" }  

capability travelRequestCapability

nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#title" hasValue "Travel Request Capability\n"
    _"http://purl.org/dc/elements/1.1#language" hasValue "English\n"
    _"http://purl.org/dc/elements/1.1#creator" hasValue "Emilia Cimpian\n"
endNonFunctionalProperties
interface travelRequestInterface

choreography

stateSignature

importsOntology{
    _"http://www.example.org/ontologies/TravelRequestOntology#"
}

in

tro#date withGrounding _"...",
tro#station withGrounding _"...",
tro#creditCard withGrounding _"..."

out

tro#time withGrounding _"...",
tro#price withGrounding _"...",
tro#ticket withGrounding _"..."

transitionRules

/*
* the invocation starts with the creation of a date instance; no condition
* need to be fulfilled in order to create this instance
*/
do
    add(_.#[
        year hasValue ?year,
        month hasValue ?month,
        day hasValue ?day
    ]memberOf tro#date)

/*
* after the date is created, the requestor creates an instance of station –
* the starting point of the trip
*/
forAll {?date} with (?date[
    ] memberOf tro#date
) do
    add(_.#[

start_Location hasValue _boolean("true"),
destination_Location hasValue _boolean("false"),
name hasValue ?name
]memberOf tro#station)
endForAll

/*
* after the instance denoting the starting point of the trip exists, the
* requestor creates an instance denoting the destination point
*/
forAll {?station} with (?station[
    start_Location hasValue _boolean("true"),
    destination_Location hasValue _boolean("false"),
] memberOf tro#station)
) do
    add(_[#[
        start_Location hasValue _boolean("false"),
        destination_Location hasValue _boolean("true"),
        name hasValue ?name
    ]memberOf tro#station)
endForAll

/*
* after all the details of the trip exist, the requestor expects the departure
* time
*/
forAll {?station} with (?station[
    start_Location hasValue _boolean("false"),
    destination_Location hasValue _boolean("true"),
] memberOf tro#station)
) do
    add(_[#[
        hour hasValue ?hour,
        date hasValue ?date
    ]memberOf tro#time)
endForAll

/*
* after the time instance was created (the departure time) the requestor
* expects the price of the trip
*/
forAll {?time} with (?time[
  ] memberOf tro#time
) do
  add(_[
    ticket_Price hasValue ?price
  ]memberOf tro#price)
endForAll

/*
* after the instance of price was created (the price of the trip) the requestor
* created the details of the credit card to be charged for the trip
*/
forAll {?price} with (?price[
  ] memberOf tro#price
) do
  add(_[
    cc_Owner hasValue ?owner,
    cc_Expiration_Date hasValue ?expirationDate,
    cc_Number hasValue ?number
  ]memberOf tro#creditCard)
endForAll

/*
* after sending the details of the credit card, the requestor
* expects the ticket
*/
forAll {?creditCard} with (?creditCard[
  ] memberOf tro#creditCard
During the communication, the requestor generates the following instances:

### Table 3: Instances Sent by the Requestor

<table>
<thead>
<tr>
<th>Instance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance travel_Date memberOf date</td>
<td>year hasValue 2005, month hasValue 12, day hasValue 20</td>
</tr>
<tr>
<td>instance cc_Expiration_Date memberOf date</td>
<td>year hasValue 2010, month hasValue 10</td>
</tr>
<tr>
<td>instance emilia memberOf person</td>
<td>name hasValue &quot;Emilia Cimpian&quot;</td>
</tr>
<tr>
<td>instance my_Cc memberOf creditCard</td>
<td>cc_Owner hasValue Emilia, cc_Expiration_Date hasValue cc_Expiration_Date, cc_Number hasValue 123456789</td>
</tr>
<tr>
<td>instance galway memberOf station</td>
<td>start_Location hasValue _boolean(&quot;false&quot;), destination_Location hasValue _boolean(&quot;true&quot;), name hasValue &quot;Galway&quot;</td>
</tr>
<tr>
<td>instance dublin memberOf station</td>
<td>start_Location hasValue _boolean(&quot;true&quot;)</td>
</tr>
</tbody>
</table>
destination_Location hasValue _boolean("false")
name hasValue ""Dublin"

Travel Provider Ontology, Service Description and Generated Instances

Table 4: Travel Provider Ontology

<table>
<thead>
<tr>
<th>Travel Provider Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsmlVariant _&quot;<a href="http://www.wsmo.org/wsml/wsml-syntax/wsml-core">http://www.wsmo.org/wsml/wsml-syntax/wsml-core</a>&quot;</td>
</tr>
<tr>
<td>namespace { _&quot;<a href="http://www.example.org/ontologies/TravelOfferOntology#%22%7D">http://www.example.org/ontologies/TravelOfferOntology#&quot;}</a></td>
</tr>
<tr>
<td>ontology _&quot;<a href="http://www.example.org/ontologies/TravelOfferOntology">http://www.example.org/ontologies/TravelOfferOntology</a>&quot;</td>
</tr>
<tr>
<td>nonFunctionalProperties</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#title">http://purl.org/dc/elements/1.1#title</a>&quot; hasValue &quot;&quot;Travel Offer Ontology&quot;&quot;</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#language">http://purl.org/dc/elements/1.1#language</a>&quot; hasValue &quot;&quot;English&quot;&quot;</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#description">http://purl.org/dc/elements/1.1#description</a>&quot; hasValue &quot;&quot;The travel offer ontology&quot;&quot;</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#creator">http://purl.org/dc/elements/1.1#creator</a>&quot; hasValue &quot;&quot;Emilia Cimpian&quot;&quot;</td>
</tr>
<tr>
<td>endNonFunctionalProperties</td>
</tr>
<tr>
<td>concept date</td>
</tr>
<tr>
<td>nonFunctionalProperties</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#description">http://purl.org/dc/elements/1.1#description</a>&quot; hasValue &quot;&quot;Date concept&quot;&quot;</td>
</tr>
<tr>
<td>endNonFunctionalProperties</td>
</tr>
<tr>
<td>year impliesType _integer</td>
</tr>
<tr>
<td>month impliesType _integer</td>
</tr>
<tr>
<td>day impliesType _integer</td>
</tr>
</tbody>
</table>

concept availability |
| nonFunctionalProperties |
| _"http://purl.org/dc/elements/1.1#description" hasValue ""Availability concept"" |
| endNonFunctionalProperties |
| available impliesType (0 1) _boolean |
concept time
  nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#description" hasValue ""Time concept"
  endNonFunctionalProperties
  hour impliesType _integer
  minutes impliesType _integer

concept person
  nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#description" hasValue ""Person concept"
  endNonFunctionalProperties
  name impliesType _string

concept price
  nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#description" hasValue ""Price concept"
  endNonFunctionalProperties
  ticket_Price impliesType _integer

concept creditCard
  nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#description" hasValue ""Credit Card concept"
  endNonFunctionalProperties
  owner impliesType person
  number impliesType _integer
  expires impliesType date

concept travel_ticket
  nonFunctionalProperties
    _"http://purl.org/dc/elements/1.1#description" hasValue ""Travel ticket concept"
  endNonFunctionalProperties
  traveler_Name impliesType _string
on_Date impliesType date
departure_Time impliesType time

custom concept route
nonFunctionalProperties
  _"http://purl.org/dc/elements/1.1#description" hasValue "Route concept"
endNonFunctionalProperties
from impliesType (0 1) _string
to impliesType (0 1) _string

Table 5: Travel Provider Service

<table>
<thead>
<tr>
<th>Travel Provider Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsmlVariant _&quot;<a href="http://www.wsmo.org/wsml/wsml-syntax/wsml-core">http://www.wsmo.org/wsml/wsml-syntax/wsml-core</a>&quot;</td>
</tr>
<tr>
<td>namespace { _&quot;<a href="http://www.examples.org/Goals/TravelOffer#">http://www.examples.org/Goals/TravelOffer#</a>&quot;,</td>
</tr>
<tr>
<td>too _&quot;<a href="http://www.examples.org/ontologies/TravelOfferOntology">http://www.examples.org/ontologies/TravelOfferOntology</a>&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>webService _&quot;<a href="http://www.examples.org/goals/travelOffer">http://www.examples.org/goals/travelOffer</a>&quot;</td>
</tr>
<tr>
<td>nonFunctionalProperties</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#title">http://purl.org/dc/elements/1.1#title</a>&quot; hasValue &quot;Travel Offer&quot;</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#language">http://purl.org/dc/elements/1.1#language</a>&quot; hasValue &quot;English&quot;</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#creator">http://purl.org/dc/elements/1.1#creator</a>&quot; hasValue &quot;Emilia Cimpian&quot;</td>
</tr>
<tr>
<td>endNonFunctionalProperties</td>
</tr>
<tr>
<td>importsOntology {</td>
</tr>
<tr>
<td>_&quot;<a href="http://www.example.org/ontologies/TravelOfferOntology">http://www.example.org/ontologies/TravelOfferOntology</a>&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>capability travelOfferCapability</td>
</tr>
<tr>
<td>nonFunctionalProperties</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#title">http://purl.org/dc/elements/1.1#title</a>&quot; hasValue &quot;Travel Offer Capability&quot;</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#language">http://purl.org/dc/elements/1.1#language</a>&quot; hasValue &quot;English&quot;</td>
</tr>
<tr>
<td>_&quot;<a href="http://purl.org/dc/elements/1.1#creator">http://purl.org/dc/elements/1.1#creator</a>&quot; hasValue &quot;Emilia Cimpian&quot;</td>
</tr>
<tr>
<td>endNonFunctionalProperties</td>
</tr>
</tbody>
</table>
interface travelOfferInterface

choreography

stateSignature

importsOntology{
    _"http://www.example.org/ontologies/TravelOfferOntology#"
}

in

too#date withGrounding "...",
too#route withGrounding "...",
too#creditCard withGrounding "...

out

too#time withGrounding "...",
too#price withGrounding "...",
too#ticket withGrounding "...

transitionRules

/*
* the invocation starts when the service receives an instance of route
*/
do
    add(_#[
        from hasValue ?from,
        to hasValue ?to
    ]memberOf too#route)

/*
* after the route is created, the service expects an instance of date – the
date of the trip
*/
forAll {?route} with (?route[
    ]memberOf too#route
) do
    add(_#[
        year hasValue ?year,
        month hasValue ?month,
day hasValue ?day
]memberOf too#date)
endForAll

/
/*
* after the date exists, the service creates an availability instance
*/
forall {?date} with (?date[
    ]memberOf too#date
) do
    add(_[#[
        available hasValue ?available
    ]memberOf too#availability)
endForAll

/
/*
* if there is an available ticket, the information about the price and time
* can be created
*/
forall {?availability} with (?availability[
    available hasValue _boolean("true"),
    ]memberOf too#availability
) do(
    add(_[#[
        hour hasValue ?hour,
        day hasValue ?day
    ]memberOf too#time),
    add(_[#[
        ticket_Price hasValue ?price
    ]memberOf too#price))
endForAll

/
/*
* after price and time have been sent, the service expects the details of
* the buyer's credit card
During the communication, the provider generates the following instances:

**Table 6: Instances Sent by the Provider**

<table>
<thead>
<tr>
<th>Instances Sent by the Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance ticket_Available memberOf availability available hasValue _boolean(&quot;true&quot;)</td>
</tr>
<tr>
<td>instance dublin_Galway memberOf travel_ticket traveler_Name hasValue &quot;Emilia Cimpian&quot;</td>
</tr>
</tbody>
</table>

```/*
forall {?price, ?time} with (?price[
    ] memberOf too#price and
    ? time[
    ] memberOf too#time
) do
    add(_#[
        owner hasValue ?owner,
        number hasValue ?number,
        expires hasValue ?expires
    ] memberOf too#creditCard)
endForAll

/*
* the tickets can be sent after the details of the credit card were received
*/
forall {?creditCard} with (?creditCard[
    ] memberOf too#creditCard
) do
    add(_#[
        traveler_Name hasValue ?name,
        on_Date hasValue ?date,
        departure_Time hasValue ?time
    ] memberOf too#travel_ticket)
endForAll
```
FM6 – 507483
Deliverable 5.5

Mapping Rules used by the Run-Time Data Mediator
During the communication, the run-time Data Mediator executes a set of previously defined mapping rules, listed in the following tables.

Table 7: Mapping Rules for the Requestor to Provider Communication

<table>
<thead>
<tr>
<th>Requestor to Provider Mapping Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mapping</strong></td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelRequestOntology#creditCard">http://www.example.org/ontologies/TravelRequestOntology#creditCard</a></td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelOfferOntology#creditCard">http://www.example.org/ontologies/TravelOfferOntology#creditCard</a></td>
</tr>
<tr>
<td><strong>classMapping</strong></td>
</tr>
<tr>
<td><strong>Mapping</strong></td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelRequestOntology#date">http://www.example.org/ontologies/TravelRequestOntology#date</a></td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelOfferOntology#date">http://www.example.org/ontologies/TravelOfferOntology#date</a></td>
</tr>
<tr>
<td><strong>classMapping</strong></td>
</tr>
</tbody>
</table>

10 The mappings are expressed in an abstract mapping language [1]
Mapping(
    http://www.example.org/ontologies/TravelRequestOntology#person
    http://www.example.org/ontologies/TravelOfferOntology#person
    classMapping( one-way person person))

Mapping(
    http://www.example.org/ontologies/TravelRequestOntology#station
    http://www.example.org/ontologies/TravelOfferOntology#route
    classMapping( one-way station route))

Mapping(
    http://www.wsmo.org/wsml/wsml-syntax#boolean
    http://www.wsmo.org/wsml/wsml-syntax#string
    classMapping( one-way Boolean string))

Mapping(
    http://www.wsmo.org/wsml/wsml-syntax#integer
    http://www.wsmo.org/wsml/wsml-syntax#integer
    classMapping( one-way integer integer))

Mapping(
    http://www.wsmo.org/wsml/wsml-syntax#string
    http://www.wsmo.org/wsml/wsml-syntax#string
    classMapping( one-way string string))

Mapping(
    http://www.example.org/ontologies/TravelRequestOntology#cc_Expiration_Date
    http://www.example.org/ontologies/TravelOfferOntology#expires
    attributeMapping( one-way
        [(creditCard) cc_Expiration_Date => date] [(creditCard) expires => date]))

Mapping(
    http://www.example.org/ontologies/TravelRequestOntology#cc_Number
    http://www.example.org/ontologies/TravelOfferOntology#number
    attributeMapping( one-way
{{{(creditCard) cc_Number => integer} {((creditCard) number => integer)}}}

Mapping
http://www.example.org/ontologies/TravelRequestOntology#cc_Owner
http://www.example.org/ontologies/TravelOfferOntology#owner
attributeMapping( one-way
{{{(creditCard) cc_Owner => person} {((creditCard) owner => person)}}})

Mapping
(http://www.example.org/ontologies/TravelRequestOntology#day
http://www.example.org/ontologies/TravelOfferOntology#day
attributeMapping( one-way {{(date) day => integer} {((date) day => integer)}}))

Mapping
http://www.example.org/ontologies/TravelRequestOntology#destination_Location
http://www.example.org/ontologies/TravelOfferOntology#to
attributeMapping( one-way
{{{(station) destination_Location => boolean} {((route) to => string)}}}
valueCondition(station {{{(station) destination_Location => boolean}}} true)

Mapping
http://www.example.org/ontologies/TravelRequestOntology#month
http://www.example.org/ontologies/TravelOfferOntology#month
attributeMapping( one-way
{{{(date) month => integer} {((date) month => integer)}}})

Mapping
http://www.example.org/ontologies/TravelRequestOntology#name
http://www.example.org/ontologies/TravelOfferOntology#from
attributeMapping( one-way
{{{(station) name => string} {((route) from => string)}}})

Mapping
http://www.example.org/ontologies/TravelRequestOntology#name
http://www.example.org/ontologies/TravelOfferOntology#name
attributeMapping( one-way
{{{(person) name => string} {((person) name => string)}}})
Mapping(
    http://www.example.org/ontologies/TravelRequestOntology#name
    http://www.example.org/ontologies/TravelOfferOntology#to
    attributeMapping( one-way [(station) name => string] [(route) to => string]))

Mapping(
    http://www.example.org/ontologies/TravelRequestOntology#start_Location
    http://www.example.org/ontologies/TravelOfferOntology#from
    attributeMapping( one-way
        [(station) start_Location => boolean] [(route) from => string])
        valueCondition(station [(station) start_Location => boolean] true)

Mapping(
    http://www.example.org/ontologies/TravelRequestOntology#year
    http://www.example.org/ontologies/TravelOfferOntology#year
    attributeMapping( one-way [(date) year => integer] [(date) year => integer]))

Table 8: Mapping Rules for the Provider to Requestor Communication

<table>
<thead>
<tr>
<th>Provider to Requestor Mapping Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping(</td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelOfferOntology#time">http://www.example.org/ontologies/TravelOfferOntology#time</a></td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelRequestOntology#time">http://www.example.org/ontologies/TravelRequestOntology#time</a></td>
</tr>
<tr>
<td>classMapping( one-way time time)</td>
</tr>
<tr>
<td>Mapping(</td>
</tr>
<tr>
<td><a href="http://www.wsmo.org/wsml/wsml-syntax#integer">http://www.wsmo.org/wsml/wsml-syntax#integer</a></td>
</tr>
<tr>
<td><a href="http://www.wsmo.org/wsml/wsml-syntax#integer">http://www.wsmo.org/wsml/wsml-syntax#integer</a></td>
</tr>
<tr>
<td>classMapping( one-way integer integer)</td>
</tr>
<tr>
<td>Mapping(</td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelOfferOntology#date">http://www.example.org/ontologies/TravelOfferOntology#date</a></td>
</tr>
<tr>
<td><a href="http://www.example.org/ontologies/TravelRequestOntology#date">http://www.example.org/ontologies/TravelRequestOntology#date</a></td>
</tr>
<tr>
<td>classMapping( one-way date date)</td>
</tr>
</tbody>
</table>
Mapping

  http://www.example.org/ontologies/TravelOfferOntology#price
  http://www.example.org/ontologies/TravelRequestOntology#price
  classMapping( one-way price price))

Mapping

  http://www.example.org/ontologies/TravelOfferOntology#travel_ticket
  http://www.example.org/ontologies/TravelRequestOntology#ticket
  classMapping( one-way travel_ticket ticket))

Mapping

  http://www.example.org/ontologies/TravelOfferOntology#hour
  http://www.example.org/ontologies/TravelRequestOntology#hour
  attributeMapping( one-way [(time) hour => integer] [(time) hour => integer]))

Mapping

  http://www.example.org/ontologies/TravelOfferOntology#minutes
  http://www.example.org/ontologies/TravelRequestOntology#minutes
  attributeMapping( one-way [(time) minutes => integer] [(time) minutes => integer]))

Mapping

  http://www.example.org/ontologies/TravelOfferOntology#day
  http://www.example.org/ontologies/TravelRequestOntology#day
  attributeMapping( one-way [(date) day => integer] [(date) day => integer]))

Mapping

  http://www.example.org/ontologies/TravelOfferOntology#month
  http://www.example.org/ontologies/TravelRequestOntology#month
  attributeMapping( one-way [(date) month => integer] [(date) month => integer]))

Mapping

  http://www.example.org/ontologies/TravelOfferOntology#year
  http://www.example.org/ontologies/TravelRequestOntology#year
  attributeMapping( one-way [(date) year => integer] [(date) year => integer])
Mapping(  
http://www.example.org/ontologies/TravelOfferOntology#ticket_Price  
http://www.example.org/ontologies/TravelRequestOntology#ticket_Price  
attributeMapping( one-way  
[((price) ticket_Price => integer) ((price) ticket_Price => integer)])

Mapping(  
http://www.example.org/ontologies/TravelOfferOntology#departure_Time  
http://www.example.org/ontologies/TravelRequestOntology#departure_Time  
attributeMapping( one-way  
[((travel_ticket) departure_Time => time) ((ticket) departure_Time => time)])

Mapping(  
http://www.example.org/ontologies/TravelOfferOntology#on_Date  
http://www.example.org/ontologies/TravelRequestOntology#departure_Date  
attributeMapping( one-way  
[((travel_ticket) on_Date => date) ((ticket) departure_Date => date)])

Log4J Logs
The following tables present the change logs for the Process Mediator and the requestor and the provider of the service.

Table 9: Process Mediator log

<table>
<thead>
<tr>
<th>Log4j logs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4247 INFO - #ProcessMediatorImpl received from Requestor: [Name = travel_Date, year hasValue 2005, month hasValue 12, day hasValue 20] mediated to: [Name = date, day hasValue 20, month hasValue 12, year hasValue 2005]</td>
<td></td>
</tr>
<tr>
<td>17508 INFO - #ProcessMediatorImpl received from Requestor: [Name = dublin, start_Location hasValue _boolean(&quot;true&quot;), destination_Location hasValue</td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>Message</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>17508 INFO</td>
<td>#ProcessMediatorImpl received from Requestor:</td>
</tr>
<tr>
<td>24810 INFO</td>
<td>#ProcessMediatorImpl received from Provider:</td>
</tr>
<tr>
<td>32994 INFO</td>
<td>#ProcessMediatorImpl received from Provider:</td>
</tr>
<tr>
<td>32994 INFO</td>
<td>#ProcessMediatorImpl received from Provider:</td>
</tr>
<tr>
<td>46235 INFO</td>
<td>#ProcessMediatorImpl received from Requestor:</td>
</tr>
<tr>
<td>46235 INFO</td>
<td>#ProcessMediatorImpl received from Requestor:</td>
</tr>
<tr>
<td>46235 INFO</td>
<td>#ProcessMediatorImpl received from Requestor:</td>
</tr>
</tbody>
</table>
mediated to: [Name = date, day hasValue 4, month hasValue 10, year hasValue 2005]]

58215 INFO - #ProcessMediatorImpl received from Provider: [Name =
  dublin_Galway, traveler_Name hasValue "Emilia Cimpian", on_Date hasValue
  [Name = travel_Date, year hasValue 2005, month hasValue 12, day hasValue
  20], departure_Time hasValue [Name = departure_Time, hour hasValue 10,
  minutes hasValue 20]]
mediated to: [Name = ticket, departure_Time hasValue [Name = time, hour
  hasValue 10, minutes hasValue 20], departure_Date hasValue [Name = date,
  day hasValue 20, month hasValue 12, year hasValue 2005]]

58215 INFO - #ProcessMediatorImpl received from Provider: [Name = travel_Date,
  year hasValue 2005, month hasValue 12, day hasValue 20]
mediated to: [Name = date, day hasValue 20, month hasValue 12, year hasValue
  2005]

58215 INFO - #ProcessMediatorImpl received from Provider: [Name =
  departure_Time, hour hasValue 10, minutes hasValue 20]
mediated to: [Name = time, hour hasValue 10, minutes hasValue 20]

Table 10: Requester’s Thread log

<table>
<thead>
<tr>
<th>Log4j logs</th>
</tr>
</thead>
</table>
| 0 INFO - #RequestorThread sending: [Name = travel_Date, year hasValue 2005,
  month hasValue 12, day hasValue 20] |
| 6631 INFO - #RequestorThread sending: [Name = dublin, start_Location hasValue
  _boolean("true"), destination_Location hasValue _boolean("false"), name
  hasValue "Dublin"] |
| 6631 INFO - #RequestorThread sending: [Name = galway, start_Location hasValue
  _boolean("false"), destination_Location hasValue _boolean("true"), name
  hasValue "Galway"] |
| 33214 INFO - #RequestorThread received: [Name = time, hour hasValue 10, minutes
  hasValue 20] |
| 33895 INFO - #RequestorThread received: [Name = price, ticket_Price hasValue 35] |
35097 INFO - #RequestorThread sending: [Name = my_Cc, cc_Owner hasValue [Name = emilia, name hasValue "Emilia Cimpian"], cc_Expiration_Date hasValue [Name = cc_Expiration, year hasValue 2010, month hasValue 10, day hasValue 4], cc_Number hasValue 123456789]

58395 INFO - #RequestorThread received: [Name = ticket, departure_Time hasValue [Name = time, hour hasValue 10, minutes hasValue 20], departure_Date hasValue [Name = date, day hasValue 20, month hasValue 12, year hasValue 2005]]

Table 11: Provider’s Thread log

<table>
<thead>
<tr>
<th>Log4j logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>17899 INFO - #ProviderThread received: [Name = route, from hasValue Dublin, to hasValue Galway]</td>
</tr>
<tr>
<td>18951 INFO - #ProviderThread received: [Name = date, day hasValue 20, month hasValue 12, year hasValue 2005]</td>
</tr>
<tr>
<td>20453 INFO - #ProviderThread sending: [Name = ticket_Available, available hasValue _boolean(&quot;true&quot;)]]</td>
</tr>
<tr>
<td>25521 INFO - #ProviderThread sending: [Name = departure_Time, hour hasValue 10, minutes hasValue 20]</td>
</tr>
<tr>
<td>25521 INFO - #ProviderThread sending: [Name = trip_Price, ticket_Price hasValue 35]</td>
</tr>
<tr>
<td>46415 INFO - #ProviderThread received: [Name = creditCard, number hasValue 123456789, owner hasValue [Name = person, name hasValue Emilia Cimpian], expires hasValue [Name = date, day hasValues {4}, month hasValues {10}, year hasValues {2005}] ]</td>
</tr>
<tr>
<td>47607 INFO - #ProviderThread sending: [Name = dublin_Galway, traveler_Name hasValue &quot;Emilia Cimpian&quot;, on_Date hasValue [Name = travel_Date, year hasValue 2005, month hasValue 12, day hasValue 20], departure_Time hasValue [Name = departure_Time, hour hasValue 10, minutes hasValue 20]]</td>
</tr>
</tbody>
</table>
ANNEX 2. WSMX PROCESS MEDIATOR JAVA DOC

Interface ProcessMediator

```java
public interface ProcessMediator
```

Process Mediation interface provides operations primarily meant for the Choreography Engine. If there is no direct matching between the choreographies of the interacting partners, then the Process Mediation is invoked by the Choreography Engine in order to reconcile the differences.

Author:

Michal Zaremba, Liliana Cabral, John Domingue, David Aiken, Emilia Cimpian, Thomas Haselwanter, Mick Kerrigan, Adrian Mocan, Matthew Moran, Brahmananda Sapkota, Maciej Zaremba

Method Detail

```java
Map<Identifier,List<Entity>> generate (Identifier sourceChoreography,
        Identifier targetChoreography, Set<Entity> data, int invocationDirection) throws
        ComponentException, UnsupportedOperationException
```

Generates a list of identifiable of the concepts/instances to be sent to any of the two involved parties

**Parameters:**

- `sourceChoreography` - the identifier of the choreography instance of the sender of a message; the choreography instance correspond to this particular phase of the conversation
- `targetChoreography` - the identifier of the choreography instance of the targeted partner; the choreography instance correspond to this particular phase of the conversation
- `data` - the information contained in the message
- `invocationDirection` - direction of the invocation (from goal to WS or the other way around)

**Returns:**

The identifier of the choreography instance of the partner that needs to receive some messages, together with the list of identifiable objects containing the actual information; each of the two partners may receive one or more messages

**Throws:**

ComponentException
java.lang.UnsupportedOperationException

Map<Identifier, List<Entity>> generate (Ontology sourceOntology, Ontology targetOntology, Set<Entity> data, int invocationDirection) throws ComponentException, UnsupportedOperationException

Generates a list of identifiable of the concepts/instances to be sent to any of the two involved parties

Parameters:

sourceOntology - the ontology instance of the sender of a message (the choreography is seen as an ontology; this parameter is not referring to the internal ontology of a partner, but to its choreography)

targetOntology - the ontology instance of the sender of a message (the choreography is seen as an ontology; this parameter is not referring to the internal ontology of a partner, but to its choreography)

data - the information contained in the message

invocationDirection - direction of the invocation (from goal to WS or the other way around)

Returns:

the identifier of the ontology (choreography) instance of the partner that needs to receive some messages, together with the list of identifiable objects containing the actual information; each of the two partners may receive one or more messages

Throws:

ComponentException

java.lang.UnsupportedOperationException

Map<Identifier,List<Entity>> generate (Choreography sourceChoreography, Choreography targetChoreography, Set<Entity> data, int invocationDirection) throws ComponentException, UnsupportedOperationException

Generates a list of identifiable of the concepts/instances to be sent to any of the two involved parties

Parameters:

sourceChoreography - the choreography instance of the sender of a message

targetChoreography - the choreography instance of the sender of a message

data - the information contained in the message

invocationDirection - direction of the invocation (from goal to WS or the other way around)

Returns:
the identifier of the choreography instance of the partner that needs to receive some messages, together with the list of identifiable objects containing the actual information; each of the two partners may receive one or more messages.

**Throws:**
- ComponentException
- java.lang.UnsupportedOperationException

---

**Interface Reasoner**

```java
public interface Reasoner

Process Mediation Reasoner interface provides a wrapper tailored to Process Mediation domain that enables reasoning on choreographies of interacting parties. As an underlying inference engine a WSML-Flight Reasoner based on KEAON2 is utilized. Currently Process Mediation Reasoner supports IF-THEN choreography constructs, which allows to model simple execution paths, not allowing more complex scenarios like parallel execution or loops. However, it provides a generic support for IF-THEN constructs and gives good foundations for the further development.

**Author:**
- Emilia Cimpian, Maciej Zaremba

---

**Method Detail**

```java
<Identifier,List<Identifier>> reasoner (Choreography targetChoreography, Set<Instance> incomingInstances)
```

Method allowing reasoning on IF-THEN target choreography. Reasoning takes into the consideration both messages that were already sent to the targeted partner but not yet delivered (because they were not expected by the partner at the given point in time) and messages that were sent and delivered. As the result a messages that can be delivered and choreography transition rules that can be fired are returned.

**Parameters:**
- `targetChoreography` - the choreography of the targeted partner
- `incomingInstances` - the instances sent to the targeted partner but not delivered yet (because they were not expected)

**Returns:**
- Map - instances (msgs.) to be sent > list of the transition rules that can be updated, and for each of this transition rules the instances that are associated with the update; null if no such transition rules exist
Interface Validator

public interface Validator

Validator is responsible for checking whether given instances (messages) are expected by targeted partner. If message is not expected by the partner it will dropped and will not be returned from the Validator.

Author:
Emilia Cimpian, Maciej Zaremba

Method Detail

Set<Entity> validate (Choreography choreography, 
    Set<Entity> instances, 
    int invocationDirection)

Parameters:
choreography - of the targeted partner for who instances are meant
instances - for which the mode needs to be checked
invocationDirection - the direction of the invocation (from goal to WS or the other way around)

Returns:
Set only the instances that are expected at some point in time by targeted partner

Interface DataStorage

public interface DataStorage

DataStorage is the entity responsible for preserving messages sent by interacting parties. It preserves the messages that were sent but not yet received by targeted partner (e.g. because not expected at given point in time or because they were not evaluated yet).

Author:
Emilia Cimpian, Maciej Zaremba

Method Detail
void `push`(Instance inst, int invocationDirection)

Stores the instance according to the direction of the invocation.

**Parameters:**
- `inst` - the instance that needs to be stored in the data storage
- `invocationDirection` - the direction of the invocation

Set< Instance> `get`(Concept concept, int invocationDirection)

Returns all instances of the given concept.

**Parameters:**
- `concept` - all instances of this concept available in the repository will be returned
- `invocationDirection` - the direction of the invocation

**Returns:**
all instances of the concept currently available in the repository

Set< Instance> `getAll`(int invocationDirection)

Returns all instances.

**Parameters:**
- `invocationDirection` - the direction of the invocation

**Returns:**
all instances currently available in the repository

void `remove`(List< Entity> instances, int invocationDirection)

Removes the list of instances.

**Parameters:**
- `instances` - the list of instances to be removed
- `invocationDirection` - the direction of the invocation
ANNEX 3. IRS-III PROCESS MEDIATOR EXAMPLE

In our approach for developing applications using Semantic Web Services with IRS-III we devise a customer team for creating Goal descriptions according to user requests and a development team for creating Web Service descriptions for the available deployed web services. The application developer is then able to create Mediator descriptions which connect domain ontologies, Goals and Web Services and provide mediation services or mapping rules for solving mismatches between ontological elements.

For illustration purposes we describe in the following the structure of WSMO descriptions (figure 6) associated with one of the goals (Assess-Equipment-Goal) defined in the case-study scenario of WP9 (e-government). This Goal describes a request for a service that can assess housing equipments (items) for a citizen who has registered for benefits within Essex County Council. Published services must find all items that suit the citizen’s situation (mobility-impairment, visual-impairment, hearing-impairment, baby-care etc) and weight, and the budget of the council’s case worker. Restrictions on the way the service can solve the goal are given by pre-conditions and post-conditions.

The Housing Department provides a composed web service (Housing-Dept-Assess-Items-WS) for solving the goal described above. The composition is defined by the orchestration in the format: (Sequence G1 G2 G3 M1 M2), where G1, G2 and G3 represent sub-goals and M1 and M2 the GG-mediators connecting them. In our example the sub-goals are: Find-Items-by-Purpose-and-Weight-Goal; Assess-Budget-Goal and Select-Suitable-Items-Goal. A third party company provides a single web service (Third-Party-Assess-Items-WS) for solving the above goal, which is described with concepts from the domain ontology Third-Party-Items-Ontology.

The mediator descriptions used in this example (fig. 6) are explained in the following. Note that all links coming from mediators connect source to target components (labels were omitted to avoid cluttering the diagram).
Fig. 6. Sample structure of WSMO descriptions for the e-government scenario

- **WG-Mediator1** – connects *Third-Party-Assess-Items-WS* to E-Gov-Assess-Items-Goal allowing it to be selected for solving the goal. This mediator defines a mediation service for converting the value of input *weight* from pounds (in the goal) to kilos (in the web-service).
- **WG-Mediator2** – connects *Housing-Dept-Assess-Items-WS* to E-Gov-Assess-Items-Goal allowing it to be selected for solving the goal. There is no mediation service and the input types are inherited from the goal.
- **OO-Mediator1** – Defines mapping rules for aligning *Third-Party-Items* domain ontology (used by the Web Service) with Equipment ontology;
- **GG-Mediator1** – Allows the output of *Find-Items-by-Purpose-and-Weight-Goal* to be used as input by *Select-Suitable-Items-Goal*.
- **GG-Mediator2** – Allows the output of *Assess-Budget-Goal* to be used as input by *Select-Suitable-Items-Goal*. It uses a mediation service to map the input type *Budget* (in the source sub-goal) to input type *Cost* (in the target sub-goal).
- **WG-Mediator3, WG-Mediator4** and **WG-Mediator5** – Connect corresponding Web Services to Sub-Goals in the orchestration. The Housing