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

Deliverable

WP 5: Service Mediation
D5.4
Business data and process-level mediation module prototype v1
Christian Drumm
John Domingue
Liliana Cabral
Adrian Mocan
Oscar Corcho
Stoyan Atanassov

June 8th, 2004
EXECUTIVE SUMMARY

Deliverable D5.4 - Business data and process-level mediation module prototype v1 is meant to be the implementation of the specification developed in deliverable D5.2 [1] which is focused on data mediation.

This document describes each software component as well as its connection to the overall dip architecture. Furthermore the interoperability of the different components is investigated.

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

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

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

**Project URL**
http://dip.semanticweb.org

**Document URL**

**EU Project officer**
Kai Tullius

**Deliverable Number**
5.4

**Title**
Business data and process-level mediation module prototype v1

**Work package Number**
5

**Title**
Service mediation

**Date of delivery**

<table>
<thead>
<tr>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>M 18</td>
<td>30-Jun-05</td>
</tr>
</tbody>
</table>

**Status**
1.0 final

**Nature**
Prototype Report Dissemination Ontology

**Dissemination Level**
Public Consortium

**Authors (Partner)**
Christian Drumm (SAP)

**Responsible Author**
Christian Drumm

**Email**
Christian.drumm@sap.com

**Phone**
+49-721-690278

**Abstract (for dissemination)**

**Keywords**

**Version Log**

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>26-01-05</td>
<td>001</td>
<td>Christian Drumm</td>
<td>Created initial table of contents</td>
</tr>
<tr>
<td>08-06-05</td>
<td>002</td>
<td>Stoyan Atanassov</td>
<td>Added Section 8 on Ontology Mapping Store</td>
</tr>
<tr>
<td>26-07-05</td>
<td>003</td>
<td>Christian</td>
<td>Incorporated reviewers comments</td>
</tr>
</tbody>
</table>
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Ireland Galway | NUIG | 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 Innovation. 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: mary@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 |
<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| 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 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sirma AI Ltd.</td>
<td>Atanas Kiryakov,</td>
</tr>
<tr>
<td></td>
<td>Ontotext Lab, - Sirma AI EAD</td>
</tr>
<tr>
<td></td>
<td>Office Express IT Centre, 3rd Floor</td>
</tr>
<tr>
<td></td>
<td>135 Tzarigradsko Chausse</td>
</tr>
<tr>
<td></td>
<td>Sofia 1784, Bulgaria</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
</tr>
<tr>
<td></td>
<td>Tel.: +359 2 9768 303</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg</td>
</tr>
<tr>
<td></td>
<td>Unicorn Solutions Ltd,</td>
</tr>
<tr>
<td></td>
<td>Malcha Technology Park 1</td>
</tr>
<tr>
<td></td>
<td>Jerusalem 96951, Israel</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
</tr>
<tr>
<td></td>
<td>Tel.: +972 2 6491111</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Pieter De Leenheer</td>
</tr>
<tr>
<td></td>
<td>Starlab- VUB</td>
</tr>
<tr>
<td></td>
<td>Vrije Universiteit Brussel</td>
</tr>
<tr>
<td></td>
<td>Pleinlaan 2, G-10, 1050 Brussel ,Belgium</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a></td>
</tr>
<tr>
<td></td>
<td>Tel.: +32 (0) 2 629 3749</td>
</tr>
</tbody>
</table>
**LIST OF KEY WORDS/ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMapLang</td>
<td>Ontology Mapping Language</td>
</tr>
<tr>
<td>OMapStore</td>
<td>Ontology Mapping Store</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Service(s)</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modelling Ontology</td>
</tr>
<tr>
<td>WSML</td>
<td>Web Service Modelling Language</td>
</tr>
<tr>
<td>WSMX</td>
<td>Web Service Execution Environment</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

**EXECUTIVE SUMMARY** ....................................................................................................................... I

**LIST OF KEY WORDS/ABBREVIATIONS** .......................................................................................... VII

**TABLE OF CONTENTS** ......................................................................................................................... VIII

1 **INTRODUCTION** ................................................................................................................................ 1

2 **MEDIATION IN WSMO** .................................................................................................................... 1

   2.1 Overall description ....................................................................................................................... 1

   2.1.1 WSMO Mediators ................................................................................................................... 2

   2.2 Data Mediation in WSMO ............................................................................................................ 2

3 **MEDIATION IN IRS** ......................................................................................................................... 2

   WG-Mediator ....................................................................................................................................... 4

   OO-Mediator ....................................................................................................................................... 5

   WW-Mediator ....................................................................................................................................... 5

   GG-Mediator ....................................................................................................................................... 6

   3.1 Installation and Usage ................................................................................................................ 6

   3.2 Requirements ............................................................................................................................... 7

4 **MEDIATION IN WSMX** .................................................................................................................... 7

   4.1 Design time tool ......................................................................................................................... 8

   4.2 Runtime component ................................................................................................................... 9

   4.3 Run-time Mediation Prototype – Technical details .................................................................. 10

   4.3.1 Current Version and Status .................................................................................................... 10

   4.3.2 Requirements ........................................................................................................................ 11

   4.3.3 Licensing ............................................................................................................................... 11

   4.3.4 Installation and Usage ............................................................................................................ 12

5 **WSMX – IRS INTEROPERABILITY** ................................................................................................. 12

6 **ONTIOLOGY-BASED INSTANCE MAPPING** .................................................................................... 12

   6.1 Status and availability ................................................................................................................. 13

7 **ONTIOLOGY MAPPING STORE** .................................................................................................... 14

   7.1 Current Version and Status ......................................................................................................... 14

   7.2 Requirements ............................................................................................................................. 14

   7.3 Licensing ........................................................................................................................................ 15

   7.3.1 Ontology Mapping Store License Agreement ........................................................................ 15

   7.3.2 Licensing of Third Party Libraries ......................................................................................... 16
7.4 Installation and Usage ................................................................. 16
7.4.1 Availability and Contacts ......................................................... 16
7.4.2 Installation of the Ontology Mapping Store Component ............. 16
7.4.3 Usage examples ....................................................................... 17
REFERENCES ................................................................................... 18
ANNEX 1 – DATA MEDIATION TESTING FRAMEWORK ......................... 19
   Requirements .................................................................................. 19
   Installation and Usage ...................................................................... 19
      Availability and Contacts ............................................................. 19
      Installation of the Ontology Mapping Store Component ................. 19
      Usage example ........................................................................... 19

LIST OF FIGURES
Figure 1a: Mediation between goal and web service. Two inputs of goal are transformed in one input of web service.......................................................... 4
Figure 2: Mediation between two sub-goals. The input of goal1 is transformed in one input of goal2.............................................................. 6
Figure 3: WSMX Data Mediation scenario ............................................ 8
Figure 4: Overview on the overall WSMX Data Mediation .................... 8
Figure 5: Overview of WSMX runtime mediation component ............... 10
1 INTRODUCTION

Deliverable D5.4 - Business data and process-level mediation module prototype v1 is intended to implement the specification presented in deliverable D5.2 [1] which is focused on data mediation. Process mediation will be covered in the next version of the prototype that will be delivered in M24 of the project. As a result of the requirements captured in D5.2, this deliverable does not just consist of one software component but of a set of components playing different roles in the overall DIP architecture. The deliverable 5.4 consists of the following parts:

- A business data mediation module based on WSMX
- A business data mediation module based on IRS III
- An ontology mapping store
- A test framework.

This document describes each software component as well as its connection to the overall dip architecture. Furthermore the interoperability of the different components is investigated. Note that WSMX and IRS III are the two reference implementation of the DIP architecture. A detailed discussion of the DIP architecture can be found here [2].

The remaining of the document is structured as follows. First section 2 briefly discusses the concept of mediation in WSMO. After that, sections 3 and 4 describe the IRS and the WSMX implementation of the data mediation module in more detail. Section 5 is focused on interoperability aspects concerning the two different implementations. Finally sections 0 and 6 describe the testing framework and the ontology mapping store.

2 MEDIATION IN WSMO

2.1 Overall description

The integration of applications is largely hindered by the mismatches between their data structure, their semantics and conceptual basis, and also by mismatches between their message exchange patterns and the protocols used for communication. In order to provide a support for resolving such inconsistencies, Web Service Modeling Ontology (WSMO) provides the concept of mediation. The primary intention of mediation in WSMO is to resolve heterogeneity problems between different WSMO resources which have been developed independently. By doing this, it intends to compensate heterogeneity between different business logics used by different business partners thereby allowing them to eventually cooperate. WSMO primarily recognizes three levels of mediation namely data-level mediation, process-level mediation, and communication protocol-level mediation. The data-level mediation is concerned with mediating data structure mismatches; the process-level mediation is concerned with mediating communication pattern mismatches; and the communication protocol-level mediation is concerned with mediating the communication protocol mismatches. However, in this deliverable we are interested in the WSMO data-level mediation. Different types of WSMO mediators are described in the following subsections.
2.1.1 WSMO Mediators

In the current version of WSMO specification, four different types of mediators are defined. They are: Ontology-to-Ontology mediators (OOMediators), Goal-to-Goal mediators (GGMediators), Web Service-to-Goal mediators (WGMediators) and Web Service-to-Web Service mediators (WWMediators). [11] states that these mediators could be the basis for resolving heterogeneities that arise at different levels such as data-level and process-level heterogeneities. These mediators are provided for allowing strict decoupling between WSMO entities, which is one of the main design principles of WSMO [12].

**OOMediators** In WSMO, OOMediator is defined to allow import of Ontologies and resolving the mismatches between them. OOMediator resolves all mismatches at Ontology level before actually importing the Ontologies be it from WSMO or non-WSMO domain.

**GGMediators** In order to allow reuse of Goals, GGMediators are defined in WSMO. It links two Goals and allows hierarchical definition of sub-goals that create a goal. The concept of refinement is hidden in the link that mediates between two heterogeneous Goals.

**WGMediators** To link between Goals and Web Services, WGMediators use Choreographies. Linking of a Goal to a Web Service implies that the Web Service fulfils the Goal, partly or fully. The WGMediator uses OOMediator and WW Mediator in order to match data and interfaces while linking the Goals and Web Services.

**WWMediators** To make heterogeneous Web Services interoperable, WSMO defines WWMediators. The WWMediators allows for Web Service composition. In addition, it can be used to facilitate process and protocol mediation [15].

2.2 Data Mediation in WSMO

The primary goal of WSMO is to provide support for seamless integration of Web Services. Web Services are heterogeneous by nature in terms of data, process, or protocol used to describe Web Services. Such heterogeneities are the major integration problem. WSMO recognizes these problems and introduces the four different types of mediators mentioned above to resolve it. However, since one of the WSMO design principle is strong decoupling, WSMO resources are built independently. Therefore, terminologies used by the WSMO resources could be different from each other. Such mismatches can be resolved by using the mediators recognized and defined by WSMO. However, the functionalities of all of these mediators boil down to the mediation of conceptual mismatches. The conceptual mismatches, as [15] states, can be resolved by use of the data mediation. Since the data mediator deals with resolving the ontological mismatches, it is central to the functionalities of WSMO mediators.

3 Mediation in IRS

This section describes data mediation in IRS-III ([http://kmi.open.ac.uk/projects/irs](http://kmi.open.ac.uk/projects/irs)), a framework and platform for developing WSMO-based Semantic Web Services. IRS-III is a powerful operational semantic framework for the representation and execution of knowledge models.
As a knowledge-based framework, IRS-III models the WSMO specification as a set of related knowledge models for the WSMO top level components (called wsmo entities). Thus, in IRS-III, Goals, Web Services and Mediators are meta-classes in corresponding ontologies. Listing 1 shows the WSMO Mediator model in IRS-III. The main class Mediator is subclassed into more specific types of mediators according to the type of source and target components.

**Listing 1. IRS-III mediation definition.**

```lisp
(defun class mediator [invokable-entity wsmo-entity]
  ((has-source-component :type wsmo-entity)
   (has-target-component :type wsmo-entity)
   (has-mediation-service :type goal)
   (has-mapping-rules :type string)))

(defun class wg-mediator (mediator)
  ((has-source-component :type (or web-service goal wg-mediator))
   (has-target-component :type (or web-service goal wg-mediator))
   (uses-mediator :type oo-mediator)))

(defun class ww-mediator (mediator)
  ((has-source-component :type (or goal web-service ww-mediator))
   (has-target-component :type (or goal web-service ww-mediator))
   (uses-mediator :type oo-mediator)))

(defun class gg-mediator (mediator)
  ((has-source-component :type (or goal gg-mediator))
   (has-target-component :type (or goal gg-mediator))
   (uses-mediator :type oo-mediator)))

(defun class oo-mediator (mediator)
  ((has-source-component :type wsmo-ontology)
   (has-target-component :type wsmo-ontology))

The Mediator concept defines semantic relations amongst a source component, a target component and a mediation service or mapping rules. Thus, mediators are transformers which provide conceptual mappings from source components to target components. IRS-III supports the implementation of Mediation Services as goals as well as the explicit declaration of mapping rules. A mediation service is implemented as a Goal and can simply be invoked for executing the web service that will transform the input into the desired output.

IRS-III maintains a library of mediators (semantic descriptions), which are created at design time using the IRS-III browser/editor. At run-time, IRS-III checks the ‘uses-mediator’ slot of WSMO components for using the mediators. In particular, when achieving a Goal or composing services, IRS-III can look up the library for mediators which connect (match) specific source and target components. The different types of WSMO mediators have been implemented according to the IRS-III framework and are described in the following subsections.
**WG-Mediator**

A WG-mediator connects a Web Service to a Goal. The source and target components can be either a Web Service or a Goal component, but can also be another wg-mediator. The IRS-III approach assumes that application developers can search for the Goal and Web Service descriptions available in a library and can provide the mediation service that can transform the inputs of an user request (Goal) into the format of inputs used by the service. A goal can be connected to many Web Services via mediators. When a user requests a goal to be achieved, the mediation service associated with the mediator of each linked web service is executed before the selection and before the invocation of the selected service.

![Figure 1a: Mediation between goal and web service. Two inputs of goal are transformed in one input of web service](image1a)

Figure 1a shows a graphical illustration of the mediation taking place between a goal and a web service via a WG-mediator. The goal requested by the application takes two inputs (first and last names), which are transformed by the mediation service into one input (name) used by the target service.

![Figure 1b: Mediation between goal and web service. One input of goal is transformed in 2 inputs of web service](image1b)

Since a mediation service can return only one output, IRS-III use a set of mediators between the goal (source) and the web service (target) in order to provide the required number of inputs to the target component as shown in figure 1b. In this example, each
medication service transforms (e.g. splits) the goal input (name) in one of the required inputs of the target component (first-name, last-name). The IRS-III engine can match the inputs and outputs for providing values as required.

**OO-Mediator**

An OO-mediator provides mediation when importing ontogies. In IRS-III, the source and target components of an oo-mediator are ontologies. Goals and Web Services and other Mediators can use the oo-mediator through the uses-mediator slot. An OO-mediator can have a mediation service associated to it for transforming inputs into outputs, otherwise it can declare explicit mappings that will be executed at runtime (figure 2). In this case, a design tool is used to support the generation of mapping rules for the ontologies involved.

![](image)

**Figure 2: Mediation between two Ontologies**

The modelling language of IRS-III (OCML) has mechanisms for mapping between entities associated with knowledge models. A simple way of associating elements of source and target components as explained above is through backward rules. Listing 2 shows how a relation mapping can be used for example to associate a concept in the domain model to an input parameter in the Goal definition. The rule above asserts the element `goal-citizen-name` with the same value as `has-client-name`. During runtime, the source and target ontologies will be merged into a new ontology related to the OO-mediator together with the relation mappings.

**Listing 2. Example of a relation mapping**

```ocml
def-relation-mapping map-citizen-name
  ((goal-citizen-name ?goal-input ?value)
   if
   (has-client-name ?citizen ?value)))
```

**WW-Mediator**

In WSMO, a WW-mediator connects two Web Services. The main use for WW-mediators is for composition of Web Services. Since in IRS-III we follow the decomposition of goals into sub-goals we replace the source and targets of a WW-
mediator by Goals. Therefore, in IRS-III sub-Goals can be composed together to solve a major Goal. A Goal will always be achieved by a selected Web Service during the invocation.

As composition of Semantic Web Services is out of the scope of this paper, we just briefly outline the role of a WW-mediator during orchestration. The provider of a Web Service describes the orchestration through control-flow mechanisms, for instance: (sequence G1 G2). The *Sequence* control command executes the given sub-goals (G1 and G2) in sequence. Figure 4 shows the graphical representation of the WW-mediator connecting G1 to G2. This mediator supports the data flow between them and the necessary mappings. By using a similar mechanism as the WG-mediator, 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 Web Service through achieve-goal are sent onto the subgoals through matching, then the associated mediators are used to connect and forward results between sub-goals providing the necessary mappings through the mediation service.

**GG-Mediator**

In WSMO, a GG-Mediator connects two goals with the purpose of specialization. For example, the goal ‘Buy Ticket’ can be specialized into ‘Buy Train Ticket’ by relating the concepts involved through inheritance into a new ontology. In IRS-III, as goals have associated ontologies, they can use mapping relations for aligning concepts between the two related goal ontologies just as OO-mediators. A GG-mediator can have a mediation service associated to it for transforming inputs into outputs, but can also declare explicit mappings, which can be generated during design time. During runtime, the source and target Goal ontologies will be merged into a new ontology related to the GG-mediator together with the mapping relations.

**3.1 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
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.

3.2 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/)

4 MEDIATION IN WSMX

In the context of Semantic Web Services the heterogeneity problems are imminent and as consequence, dynamic and flexible solutions are required. At this stage, Web Service Execution Environment (WSMX) offers full support for data mediation [7], and the conceptual foundations for process mediations [8]. An implementation for process mediation will follow in the near future as well.

The main focus in WSMX data mediation is towards instance transformation, aiming to enable seamless communication between various business partners from the data representation point of view. That is, the envisioned scenario implies the existence of two business partners willing to communicate one with each other. The problems to be solved appears when the two partner use different representation for their data, and none of them can or wants to adjust their information system to accommodate this heterogeneity problem. The cause of the unwillingness or of the impossibility to change the ontologies and/or the working practice might be the cost as well as other factors as time constraints, contractual obligations with other partners etc. None of them are covered in this document because they are not relevant for the description of a particular prototype, but details of such analysis can be found in [17] and [18]. Such situation would occur very often in a dynamic environment as the one envisioned by the Semantic Web. In this context, an assumption we take is that both of the communicating parties are using ontologies to describe their data. By this, the data to be mediated is represented by ontology instances that have to be transformed from the terms of the source ontology to the terms of the target ontology (see Figure 3). Please note that the mediator component might consider as target ontology not directly the ontology used by the target party, but an ontology understandable by it by the means of, for example, another mediator system (symbolized by a cloud in our picture). This process we call
instance transformation and the component able to perform these transformations is described in the following subsections.

Figure 3: WSMX Data Mediation scenario

The data mediation process in WSMX consists of two main phases: design time phase and run time phase. WSMX offers implementation for both of these two phases: a design time tool and a runtime tool. Even though only the run-time component is part of the WSMX architecture (and run time components are the focus of this deliverable) we provide a short description of the WSMX design-time component due to its tight relation with the run time component. An overall view of the data mediator is provided in Figure 4).

Figure 4: Overview on the overall WSMX Data Mediation

4.1 Design time tool

The data mediation design time tool is not part of the WSMX architecture but has a crucial role in the effectiveness and the quality of the results delivered by the run-time component as shown in Figure 4. In fact, the design time component is an ontology mapper, a graphical user interface supporting semi-automatic ontology alignment in an interactive mode. This means that the human user is kept in the loop and their inputs are
required during the mapping process in order to make certain choices and validations. The ontologies that can be loaded by the mapper are WSMO ontologies, represented in Web Service Modelling Language (WSML) [5]. For other kind of ontologies represented in different languages, an adapter could be used to enable their loading in the mapping tool.

The output of the design-time tool is an alignment of the two ontologies (the source and the target ontology) represented by a set of mappings between entities from the source ontology and entities from the target ontology. The mappings considered for this version of the prototype are unidirectional mappings, and symmetric mappings between the target and the source ontologies are required when bidirectional alignments are needed. The mappings are represented in the abstract mapping language described in [4] and handled by using the Ontology Mapping Language API (see section 7.2 for more details); they are stored in an external storage for further refinements or for the use of a runtime component. For this version a relational database is used (MySQL1) but in principle any other persistent storage can be used (e.g. the Ontology Mapping Store2 described in section 7).

4.2 Runtime component

The runtime mediation component is part of the WSMX architecture and its task is the one depicted in Figure 3: to perform the actual transformation of the source data (source ontology instances) to the format required by the target party (target ontology instances). The whole process is an automatic one, no human intervention being required in any stage of these transformations. The main drawback is that the results and the effectiveness of this component (in fact its capability of performing the mediation of incoming data) directly depends on the collection of mappings existing in the storage. On the other hand, relying on existing mappings, checked and validated by a human user, we can ensure 100% accuracy of the mediation process, absolutely necessary in a business context.

As we mentioned in the previous subsection the mappings are represented using the abstract mapping language presented in [5]. As a consequence we can distinguish two main tasks that are performed by the runtime component. The first task is the so called grounding of the abstract mapping language to a concrete ontology representation language and the second task is to perform the actual data transformation and to deliver the target data in the expected format.

The grounding mechanism has the role of associating formal semantics to the abstract mapping language and basically, to offer transformations a concrete language for which reasoning facilities are available. Even if WSMX data mediation acts in the context of WSMO ontologies and the default ontology representation language is WSML, we chose to provide and use a grounding mechanism in Flora-23 [7]. This decision was made due to the fact that there is no stable WSML reasoner implementation yet, and because Flora-2 is good candidate for a WSML reasoner [1].

1 See http://www.mysql.com/ for more details.
2 At the time of writing this document the usage of the Ontology Mapping Store in conjunction with the WSMX runtime data mediator is under investigation.
3 See http://flora.sourceforge.net/ for more information
The data transformation phase takes place in the Flora-2 environment by applying the grounded mapping rules to the incoming source ontology instances. After the mapping rules are evaluated the results (target ontology instances) are retrieved and made available for further computations to the requester of the mediation process (see Figure 5).

**Figure 5: Overview of WSMX runtime mediation component**

The implementation of the grounding mechanism is part of the runtime component (*Flora Rules Generator* modules) in order to allow the connection of different repositories containing abstract mappings. These repositories could become an important point of synchronization between various mediation systems, offering alignments between ontologies without committing to any ontology language. In addition, storing the mappings in an abstract form offers significant advantage in the context of evolving ontologies when mappings have to be maintained and updated. At the same time, the runtime components might offer a caching mechanism to reduce the overload of applying the grounding each time for the same request types – this features are to be offered in one of the next versions of the WSMX runtime component as well.

### 4.3 Run-time Mediation Prototype – Technical details

#### 4.3.1 Current Version and Status

The current version of the run-time mediator is able to use the mappings stored in external storage and to execute them against the incoming instances. The tool offers the Java interfaces presented in Table 1. The storage contains mappings as specified in [4], and it uses the Ontology Mapping Language API\(^4\) as an in-memory object model to represent them.

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>mediate(Ontology sourceOntology, Ontology targetOntology, Set&lt;Identifiable&gt; data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map&lt;Identifiable, List&lt;Identifiable&gt;&gt;</td>
<td>Transforms a set of source ontology instances into instances of the target ontology.</td>
</tr>
</tbody>
</table>

| List<Identifiable> | mediate(Ontology sourceOntology, Ontology targetOntology, |

\(^4\) See [http://www.omwg.org](http://www.omwg.org) for more details.
Transforms a given source ontology instance into instances of the target ontology.

By invoking the first two methods the data to be mediated (i.e. instances) have to be provided in terms of WSMO API objects and the mediated data are returned accordingly, as WSMO API objects. The third method accepts the data to be mediated in form of text messages (WSML documents) and returns the mediated data as WSML documents as well.

For this version the tool uses a relational database storage that can be populated using the mapping tool integrated in Web Service Modelling Toolkit (WSMT) [11]. For further versions the tool will be integrated with the ontology mapping storage described in Section Error! Reference source not found..

4.3.2 Requirements

The Run-time mediation tool has the following requirements:

- **Platform**: Java 1.5
- **External systems**: XSB/Flora-2
- **External Libraries**:
  - WSMO API and WSMO4J
  - Ontology Mapping Language API
  - Interprolog, a Java – XSB Prolog interface

4.3.3 Licensing

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

---

5 See http://xsb.sourceforge.net/
6 See https://sourceforge.net/projects/wsmo4j
7 See http://www.declarativa.com/interprolog/
4.3.4 Installation and Usage

The run-time component is designed to be part of the WSMX architecture, and it offers well-defined interfaces showing how it can be invoked (see Table 1). As a consequence it can be downloaded together with WSMX from [https://sourceforge.net/projects/wsmx](https://sourceforge.net/projects/wsmx).

Outside the WSMX architecture the run-time component can be used in the following ways:

- As a standalone application able to connect to the provided mapping storage and to perform mediation of instances provided as WSML documents. It offers a small graphical interface where the user can set the source and the target ontology, provide the data to be mediated and retrieve the mediated data.

- As a deployed Web Service that can be invoked with the source and target ontology IDs and data to be mediated and that returns the mediated data. The mapping tool integrated in WSMT\(^8\) can directly invoke such Web Services by using the WSMT Invoker. Another option would be to provide a web page as a front end for the run-time mediator. The WSDL file of a deployed run-time mediator as a Web Service is available at: [http://140.203.154.164/wsmx/services/RuntimeMediator?wsdl](http://140.203.154.164/wsmx/services/RuntimeMediator?wsdl).

5 WSMX – IRS INTEROPERABILITY

- The interoperability between WSMX and IRS-III mediation components will be realized by complying with the DIP API. The DIP architecture is designed for allowing the plug-and-play of different component implementations. These components can also be delivered as Web Services.

- Moreover, IRS-III will provide a translator between the underlying languages WSML and OCML.

6 ONTOLOGY-BASED INSTANCE MAPPING

The specification of the mediation module (D5.2 [1]) clearly states the requirements for ontology-based instance mapping. A detail description of its functionality and relation to the other components can be found in this document. Therefore we focus in this section only on the description of the prototypical implementation.

The main objective of ontology-based instance mapping is to automatically mediate instance data in a service by using a central ontology. This is illustrated in the figure below for an “order” service.

\(^8\) Web Service Modeling Toolkit (WSMT) downloadable from: [https://sourceforge.net/projects/wsmx](https://sourceforge.net/projects/wsmx). More details about WSMT you can find at: [http://www.wsmo.org/TR/d9/d9.1/v0.2/](http://www.wsmo.org/TR/d9/d9.1/v0.2/)
Script 1 has to be written once. It generates Script 2 from the ontology used to generate the template for Script 3 (to be filled manually).

Script 5 has to be written once to generate the resulting runtime Instance Mapping 6 from the schema-to-ontology Scripts 3 and 4 (lifted from the XML Schemas describing the instances). The scripts use XSLT and can also include Java calls.

Script 2 generates the XSLT script templates 3 and 4 that have to be filled manually by using the ontology.

The generated Script 6 is used for the instance mapping at run-time and could use Java calls to access ontology reasoning, if this is needed.

6.1 Status and availability

The current version of the mapping workflow is a prototypical 0.1 with some restrictions. Complex structure changes are not possible at the moment. The workflow only works with some given schemata and has a floating API. It is available as a Web Service. The describing WSDL can be found at:


The following code extract from the WSDL file shows which inputs the translation service expects:

```
<wSDL:message name="translate">
  <wsdl:part name="instance" type="xsd:string"/>
  <wsdl:part name="schemaNoFrom" type="xsd:integer"/>
  <wsdl:part name="schemaNoTo" type="xsd:integer"/>
</wSDL:message>
```
The 'instance' is the instance which will be translated. Therefore the string contains a XML document which is an instance of the schema with the id 'schemaNoFrom'. The response of the function 'translate' is a translation of the given XML document into an instance of the schema with the id 'schemaNoTo'. The schema numbers are internal numbers to identify the schemas in the list of schemas, we currently support.

7 ONTOLOGY MAPPING STORE

The ontology mapping store (OMapStore) allows for storage and retrieval of ontology mappings. An ontology mapping is a formal specification of a correspondence between two ontologies, e.g. equivalent classes and relationships. One of the main purposes of the ontology mappings is to allow for interoperability between programs using data represented with respect to different ontologies. This could be achieved through a variety of ontology mediation strategies, e.g. instance transformation, query re-writing.

The OMapStore is designed to support mappings represented according to the Ontology Mapping Language API\(^9\) (OMapLang). The latter is covering the following functionality (i) in-memory structures for representation of mapping expressions; (ii) parsing and serialization of mapping expressions.

The OMapStore is taking care of:

- In-memory model allowing for representation of ontology mappings in terms of: id, name, source and target ontologies, description, version. The definitions of the mappings are only managed as strings, which could be parsed and managed further with the OMapLang.
- Storage of mappings - when a mapping is stored, the store generates and ID.
- Retrieval of mappings - through a variety of restrictions on its attributes.

7.1 Current Version and Status

The current version of OMapStore is 0.2. It implements the basic functionalities identified as requirements in the scope of DIP and integrates with all the other relevant packages – see required libraries (OMWG, SDK). OMapStore can be expected to develop further based on the feedback after the first round of integration.

The immediate development plans include integration with the forthcoming versions of wsmo4j (to support WSML 0.2 and WSMO 1.1) and the Ontology Mapping Language API.

7.2 Requirements

**Nature:** A Java library without any user interface.

**Interfaces (API, Web Services):** a Java API.

**Platform:** JDK 1.4.2 or 1.5.

\(^9\) http://www.omwg.org/tools/OntologyMappingLanguage/
**Supported standards:** the Ontology Mapping Language presented in [9] and supported by the OMapLang (see below).

**Required Libraries (OMWG\(^{10}\), SDK Cluster\(^{11}\), WSMO\(^{12}\)-related):**

**Ontology Mapping Language API** (OMapLang) is library allowing for in-memory representation, management, serialization and parsing of mapping expressions. OMapLang is on its own dependent on wsmo4j (see below). OMapLang is integrated with OMapStore in order to allow for parsing and further management of mapping definitions. The main functionality of the store is independent from the representations of the definitions. The version of OMapLang used in the current version of OMapStore is the one from 17/05/2005.

**wsmo4j\(^{13}\)** is an API and a reference implementation for building Semantic Web Services applications compliant with the Web Service Modeling Ontology (WSMO). It is used within OMapStore for only for the purpose of Ontologies (as a high-level object). In a sense OMapStore has a rather weak dependency on wsmo4j - it can easily be removed with any other representation of ontologies where those are identified by an URI. The version of wsmo4j used in the current version of OMapStore is 0.3.2, compliant with WSMO v.1.0.

**Required Libraries (others):**

**Lucene\(^{14}\)** – an open-source Information retrieval (IR) engine from the Apache Foundation. Within OMapStore it is used for mapping storage and retrieval. The version of Lucene used in the current version of OMapStore is 1.4.3.

**JUnit\(^{15}\)** – an open-source simple framework for writing and running automated tests. Within OMapStore it is used for testing of the example scenarios. The version of JUnit used in the current version of OMapStore is 3.8.1

### 7.3 Licensing

#### 7.3.1 Ontology Mapping Store License Agreement

Copyright (c) 2005, Ontotext Lab, Sirma.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

---

\(^{10}\) [http://www.omwg.org/](http://www.omwg.org/)


\(^{12}\) [http://www.wsmo.org/](http://www.wsmo.org/)

\(^{13}\) [http://wsmo4j.sourceforge.org/](http://wsmo4j.sourceforge.org/)

\(^{14}\) [http://lucene.apache.org/java/docs/index.html](http://lucene.apache.org/java/docs/index.html)

\(^{15}\) [http://www.junit.org/](http://www.junit.org/)
Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

7.3.2 Licensing of Third Party Libraries

Licensing of third party libraries and components required for OMapStore:

**Ontology Mapping Language API** - (c) Copyright DERI®. It is an open-source library.

**wsmo4j** - (c) Copyright Ontotext Lab, Sirma. It is an open-source library, available under the same LGPL conditions.

**Lucene** – (c) Copyright Apache Corporation. This PRODUCT includes software developed by the Apache Software Foundation (http://www.apache.org/). The use of Lucene within the PRODUCT complies with the proprietary licence agreement for Lucene – the Apache Software License, Version 1.1, following verbatim below. Any redistribution or modification of the PRODUCT, according to the provisions, terms and conditions of this Licence, should also conform to the terms and conditions in the Apache Software License, Version 1.1.

**JUnit** - (c) Copyright JUnit.org. It is a regression testing framework written by Erich Gamma and Kent Beck. It is used by the developer who implements unit tests in Java. JUnit is Open Source Software, released under the Common Public License Version 1.0. Your use of JUnit 3.8.1 is subject to the terms and conditions of the Common Public License, Version 1.0.

7.4 Installation and Usage

7.4.1 Availability and Contacts

**Version:** 0.2, 02 June 2005.

**Download:** [http://www.omwg.org/tools/omapstore/v0.2/omapstore.zip](http://www.omwg.org/tools/omapstore/v0.2/omapstore.zip)

**Source control:** To be made available from CVS of the DOME SourceForge project.

**Contact person:** Stoyan Atanassov, stoyan@sirma.bg

7.4.2 Installation of the Ontology Mapping Store Component

The OMapStore is distributed as a ZIP archive, which should be extracted in a separate folder. The archive file is originally named omapstore.zip and has the following contents:

- FactSheet.html - this document;
- doc folder - contains Javadoc documentation for this mapping component;
- lib folder - contains all the necessary libraries (jar files);
- src folder - contains all source files;
- example.txt - sample definition of a mapping in the mapping language format;
- mappingstore.jar - the OMapStore component provided as a Java library. For integration in Java programs, one needs this file plus the ones in the lib folder to be in the CLASSPATH.
7.4.3 Usage examples

Five simple scenarios illustrating the functionality of the Mapping store component (for store, load, remove, simple and complex search of mappings) are available in "src\org\omwg\mediation\mappingstore\examples" directory.

SearchByNameScenario - creates two mappings from string definitions, stores them and makes simple search by given prefix of mapping name.

ComplexSearchScenario - creates two mappings from string definitions, stores them and makes complex search by description keywords and mapping name prefix

SearchByTargetScenario - creates a mapping, using a parser (from the OMapLang library) to parse a definition from a file; next stores it and makes simple search by given target ontology prefix name.

SearchByVersionScenario - creates some mappings generated from string definitions, stores them and makes simple search by version.

EditMappingScenario - creates a mapping, stores and loads it. After that edits mapping description and updates the version.
REFERENCES


ANNEX 1 – DATA MEDIATION TESTING FRAMEWORK

The Data Mediation Testing Framework is a universal tool to test Web Services. It is tailored towards testing the implementation of data mediation Web Services. The testing framework was motivated by the observation that in order to effectively develop mediators and mediation services, a testing facility was necessary. Therefore the testing framework was developed with 2 possible usage scenarios in mind:

1. The debugging of mediator service implementations
2. The debugging of mapping rules.

However, as the data level mediation modules will provide their functionality as a Web Service both usage scenarios can easily be realized with a tool capable of invoking a Web Service and comparing the expected and the actual response of the service. Therefore the Data Mediation Testing Framework was developed as a tool that is able to invoke Web Services and compare the expected and the actual response of the invoked service. If the expected and the actual result are identical, the test is passed. Otherwise the test fails and a detailed description of the difference between expected and actual response is provided.

Requirements

Nature: A Web Application Archive deployable in every standard conform servlet engine.

Interfaces (API, Web Services): none provided

Platform: JDK 1.4.* or 1.5.

Servlet-Container: The tool needs a servlet engine conforming to the Java Servlet Specification version 2.3 (e.g. Apache Tomcat 5.0)

Installation and Usage

Availability and Contacts

Version: 1.0.

Download: http://www.sdn.sap.com (The tool can be found in the downloads section

Contact persons:  Christian Drumm, christian.drumm@sap.com
Harald Fuchs, harald.fuchs@sap.com

Installation of the Ontology Mapping Store Component

The installation of the tool depends mainly on the servlet container in use. For information regarding the installation of WAR files refer to the documentation of the particular servlet engine.

Usage example

As described before the Data Mediation Testing Framework is a tool for testing Web Services. You have to specify some calling arguments for the service and an array of pairs comprising a parameter set and the expected response. For each pair the tool will construct a SOAP message, send it to the service, and check to see if the reply matches the expected response.
The following calling arguments must be supplied:

- The service URL
- The service URN
- The name of the operation to call
- The SOAPaction HTTP header field (default: empty string)

The exact parameters you have to supply here depend on the SOAP engine of the service you want to test. Some engines insist on the presence of such an HTTP header (regardless of its value), some don't care at all.

The parameter set of a request/response pair have to be supplied in the form of an XML fragment. Since this fragment usually references XML namespaces to be declared in the SOAP envelope, you also have to supply a mapping between namespaces and their identifiers (the default provided is fine for many services).

Since the request/response pairs can get quite large, you can supply them either by typing them into a text area or by uploading a file you created earlier with your editor. The following example will show how the Data Mediation Testing Framework can be used to test Web Services.

Say you have an Apache Tomcat servlet container running on your host, along with an Apache Axis SOAP engine. This engine by default provides a Web Service telling you the Axis version. The calling arguments look like that:

**URL:** http://localhost:8080/axis/services/Version

**URN:** urn:Version

**Operation:** getVersion

The parameter set of the request is empty because the request consists just of the operation name. The expected response will look similar to the following XML fragment (depending on your Axis version):

```xml
<getVersionReturn
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="soapenc:string">
 Apache Axis version: 1.2 Built on May 03, 2005 (02:20:24 EDT)
</getVersionReturn>
```

The SOAP request constructed by the tool will be the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://schemas.xmlsoap.org/soap/encoding"
 soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <soap:Body>
  <dummy_nsid_generated_by_dipmtf:getVersion
 xmlns:dummy_nsid_generated_by_dipmtf="urn:Version">
  </dummy_nsid_generated_by_dipmtf:getVersion>
 </soap:Body>
</soap:Envelope>
```
Axis will reply to this request with the following response:

```xml
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <ns1:getVersionResponse
            soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:ns1="urn:Version">
            <getVersionReturn xsi:type="soapenc:string"
                xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"/>
            Apache Axis version: 1.2 Built on May 03, 2005 (02:20:24 EDT)
        </ns1:getVersionReturn>
    </soapenv:Body>
</soapenv:Envelope>
```

The tool strips the envelope and the top XML node of the body and compares the rest literally with what you supplied as an expected response.

There are two additional features to mention: ignoring element names and regular expressions.

Some SOAP engines (e.g. Perl SOAP::Lite) don't bother the user with the need to invent meaningless XML element names; if you don't supply them, the engine will invent them for you. A response might look like that:

```xml
<s-gensym3 xsi:type="xsd:string">Perl</s-gensym3>
<s-gensym5 xsi:type="xsd:string">C</s-gensym5>
<s-gensym7 xsi:type="xsd:string">sh</s-gensym7>
```

Since you can't predict the gensym numbers in advance, the tool provides a checkbox which allows you to ignore element names altogether when comparing expected and actual response. You still need to supply valid tag names since the expected response must be well-formed XML, but a tag name mismatch won't be flagged as an error any more.

The second feature comes handy when you don't know the exact values your service will return. For example, if you want to test a service returning currency exchange rates, you know that you will get back a floating-point number, but you don't know the exact rate currently valid. In order to test a service like that, you can paraphrase the return value by means of a regular expression. When a value begins and ends with '+' signs (octal 0261 in ISO Latin-1), the string between them is interpreted as a regexp according to java.util.regex.

In the case of a floating-point number, you would thus use something like '±^\d+\.|\d+$±'. The following XML fragments shows a request using exactly this feature and a response that would be accepted:

- **Expected response:**
  ```xml
  <result xmlns=""">±^\d+\.|\d+$±</result>
  ```

- **Actual response:**
<result>30</result>