



Data, Information and Process Integration  
with Semantic Web Services

**DIP**

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

**FP6 - 507483**

Deliverable

**WP 6: Interoperability and Architecture**

**D6.10**

**Architecture Prototype V2**

Thomas Haselwanter

Bernhard Schreder

Alexander Wahler

Michal Zaremba

Aleksandar Balaban

December 19<sup>th</sup>, 2005





## SUMMARY

This deliverable provides an overview of the second version of the DIP architecture prototype including detailed installation guidelines and documentation. The prototype itself consists of the WSMX core architecture as well as several additional components.

WSMX [14] is an execution environment which enables discovery, selection, mediation, invocation and interoperation of the Semantic Web Services (SWS). The mission and ultimate goal of the WSMX working group is to define a SWS architecture and build a fully fledged enterprise application based on the conceptual model of WSMO [13]. WSMX is based on the conceptual model provided by WSMO, being at the same time a reference implementation of it. It is the scope of WSMX to provide a test bed for WSMO and to prove its viability as a mean to achieve dynamic interoperability of Semantic Web Services. In this document we report on the open source implementation of the system, which is used as a reference implementation of the DIP architecture.

Since July 2004, the code base of WSMX is hosted at SourceForge – the world’s largest repository of open source projects. The WSMX open source implementation can be accessed at <http://sourceforge.net/projects/wsmx/>, where both current and previous releases, as well as all the code are available.

This deliverable contributes to the Open Source Semantic Web Service Architecture and is relevant to all components developed in DIP. The target audiences of this deliverable are developers and IT experts who are interested to test and evaluate the prototype.

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

<b>IST Project Number</b>	FP6 – 507483	<b>Acronym</b>	DIP
<b>Full title</b>	Data, Information, and Process Integration with Semantic Web Services		
<b>Project URL</b>	<a href="http://dip.semanticweb.org">http://dip.semanticweb.org</a>		
<b>Document URL</b>			
<b>EU Project officer</b>	Kai Tullius		

<b>Deliverable</b>	<b>Number</b>	6.10	<b>Title</b>	Architecture Prototype V2
<b>Work package</b>	<b>Number</b>	6	<b>Title</b>	Interoperability and Architecture

<b>Date of delivery</b>	<b>Contractual</b>	M24	<b>Actual</b>	9-Jan-06
<b>Status</b>	version. 0.04		final <input checked="" type="checkbox"/>	
<b>Nature</b>	Prototype <input checked="" type="checkbox"/> Report <input type="checkbox"/> Dissemination <input type="checkbox"/> Ontology <input type="checkbox"/>			
<b>Dissemination Level</b>	Public <input checked="" type="checkbox"/> Consortium <input type="checkbox"/>			

<b>Authors (Partner)</b>	Thomas Haselwanter (UIBK), Bernhard Schreder (NIWA), Alexander Wahler (NIWA), Michal Zaremba (NUIG), Aleksandar Balaban (NIWA)			
<b>Responsible Author</b>	Bernhard Schreder		<b>Email</b>	schreder@niwa.at
	<b>Partner</b>	NIWA	<b>Phone</b>	+4313195843-13







<b>Abstract (for dissemination)</b>	This deliverables provides an overview of the second version of the DIP architecture prototype including detailed installation guidelines and documentation.
<b>Keywords</b>	Execution environment, architecture, semantic web, semantic web services, execution semantics, prototype








Version Log			
Issue Date	Rev No.	Author	Change
03-dec-05	001	Bernhard Schreder	Initial version
17-dec-05	002	Alexander Wahler	Added summary and conclusion
19-dec-05	003	Bernhard Schreder	Review version
09-jan-06	004	Bernhard Schreder	Updates based on reviewers' comments




<b>Reviewer</b>	
-----------------	--

	Vassil Momtchev	<b>Email</b>	vassil.momtchev@ontotext.com
	<b>Partner</b> Ontotext	<b>Phone</b>	
	Mick Kerrigan	<b>Email</b>	michael.kerrigan@deri.org
	<b>Partner</b> NUIG	<b>Phone</b>	

## Project Consortium Information

Partner	Acronym	Contact
National University of Ireland Galway	NUIG  National University of Ireland, Galway <i>Ollscoil na hÉireann, Gaillimh</i>	Dr. Sigurd Harand Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland Email: <a href="mailto:sigurd.harand@deri.org">sigurd.harand@deri.org</a> Tel: +353 91 495112
Fundacion De La Innovacion.Bankinter	Bankinter 	Monica Martinez Montes Fundacion de la Innovacion. BankInter Paseo Castellana, 29 28046 Madrid, Spain Email: <a href="mailto:mmtnez@bankinter.es">mmtnez@bankinter.es</a> Tel: 916234238
Berlecon Research GmbH	Berlecon 	Dr. Thorsten Wichmann Berlecon Research GmbH Oranienburger Str. 32 10117 Berlin, Germany Email: <a href="mailto:tw@berlecon.de">tw@berlecon.de</a> 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: <a href="mailto:john.nj.davies@bt.com">john.nj.davies@bt.com</a> 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 : <a href="mailto:Karl.Aberer@epfl.ch">Karl.Aberer@epfl.ch</a> Tel: +41 21 693 4679
Essex County Council	Essex  Essex County Council	Mary Rowlett, Essex County Council PO Box 11, County Hall, Duke Street Chelmsford, Essex, CM1 1LX United Kingdom. Email: <a href="mailto:maryr@essexcc.gov.uk">maryr@essexcc.gov.uk</a> Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI 	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany Email: <a href="mailto:abecker@fzi.de">abecker@fzi.de</a> Tel: +49 721 9654 0
Partner	Acronym	Contact

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

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

**LIST OF KEY WORDS/ABBREVIATIONS**

ASM	Abstract State Machine
GUI	graphical user interface
IDE	Integrated Development Environment
SOA	Service Oriented Architecture
SWS	Semantic Web Services
TUI	text user interface
WSMO	Web Services Modeling Ontology
WSMX	Web Services Execution Environment

---

## TABLE OF CONTENTS

<b>SUMMARY</b> .....	<b>I</b>
<b>LIST OF KEY WORDS/ABBREVIATIONS</b> .....	<b>VII</b>
<b>TABLE OF CONTENTS</b> .....	<b>VIII</b>
<b>1 INTRODUCTION</b> .....	<b>1</b>
<b>2 ARCHITECTURE PROTOTYPE FACT SHEET</b> .....	<b>1</b>
2.1 Description of purpose, scope and functionality .....	1
2.2 Available Release and Components .....	1
2.3 Installation Guidelines .....	3
2.3.1 WSMX Software Prerequisites.....	3
2.3.2 WSMX Installation Instructions .....	3
2.4 API information.....	4
2.5 Licence information.....	4
2.6 How to use the prototype.....	4
2.7 Roadmap for future plans .....	4
<b>3 ARCHITECTURE PROTOTYPE DOCUMENTATION</b> .....	<b>5</b>
3.1 Server Administration GUI .....	5
3.2 WSMX Component descriptions.....	6
3.2.1 Adapter Framework.....	6
3.2.2 Communication Manager .....	7
3.2.3 Resource Manager .....	8
3.2.4 Run-time Data Mediator.....	8
3.2.5 Process Mediator .....	9
3.2.6 Choreography Engine.....	11
3.2.7 WSML Flight Reasoner.....	11
3.3 Additional documentation .....	12
<b>4 CONCLUSION</b> .....	<b>12</b>
<b>REFERENCES</b> .....	<b>14</b>

## LIST OF FIGURES

Figure 1: WSMX Management Console – Main View .....	5
Figure 2: WSMX Management Console – Server View .....	6
Figure 3: WSML Flight Reasoner .....	12

**LIST OF TABLES**

Table 1: WSMX packages .....	2
Table 2: Real WSMX components .....	2

## 1 INTRODUCTION

This deliverable provides the documentation for the second version of the DIP architecture prototype, representing an implementation of the architecture first specified in [1], revised in [4] and last updated in [11]. The prototype itself consists of the updated WSMX core architecture as well as a growing collection of additional components. This document is itself an updated version of the deliverable describing the last prototype ([12]), and thus structured in a similar way: Section 2 consists of the Architecture Prototype Fact Sheet, containing all the relevant information concerning the prototype, i.e. updated installation guidelines. Section 3 contains descriptions of the currently available components for the DIP architecture, as well as the links to additional documentation on used APIs. Finally, the conclusion summarises the status of the prototype architecture.

## 2 ARCHITECTURE PROTOTYPE FACT SHEET

The contact person for the architecture prototype is Michal Zaremba (michal.zaremba@deri.org)

### 2.1 Description of purpose, scope and functionality

WSMX is an execution environment which enables discovery, selection, mediation, invocation and interoperation of Semantic Web Services (SWS). The mission and ultimate goal of the working group is to define a SWS architecture and build a fully fledged enterprise application based on the conceptual model of WSMO. WSMX is based on the conceptual model provided by WSMO, being at the same time a reference implementation of it. It is the scope of WSMX to provide a test bed for WSMO and to prove its viability as a means of achieving dynamic interoperability of Semantic Web Services. The functionalities of WSMX are divided into two main categories: the enterprise system features of the framework and the component functionalities. These basic functionalities have not changed since the last prototype and are described in more detail in [12].

Since July 2004 the code base of WSMX is hosted at SourceForge – the world’s largest repository of open source projects. The WSMX open source implementation can be accessed at <http://sourceforge.net/projects/wsmx/>, where both current and previous releases, as well as all the code are available.

Additionally the website <http://www.wsmx.org/> hosts nightly builds of the WSMX Core and Components from the WSMX CVS, as well as a FAQ and additional documentation.

### 2.2 Available Release and Components

The current release of WSMX (version 0.2) is available in the form of different packages. A description of the packages and the included components and libraries is shown in Table 1:

**Table 1: WSMX packages**

Package name	Description
wsmx_lite-0.2-src.zip	sources without third party libraries
wsmx_all-0.2-src.zip	sources with third party libraries
wsmx-thirdparty-0.2.zip	necessary third party libraries
wsmx_core_and_mockups-0.2-bin.zip	compiled version of core engine and mockup components
wsmx_core_with_branch_of_execution_semantics-0.2-bin.zip	compiled version of core engine with one working branch of execution semantics
wsmx_components-0.2-bin.zip	compiled components
wsmx-integration-API-0.2.1b.zip	Contains the Integration API and corresponding javadoc

The packages include a number of mock-up components as described in [12]; in addition to the mock-up components a growing number of fully implemented components are available with the current release of WSMX. Table 3 shows the components available from either the WSMX packages or the WSMX CVS. A detailed description of these components can be found in section 3 of this deliverable, as well as in [11].

**Table 2: Real WSMX components**

Component name
Adapter Framework
Communication Manager
Resource Manager
Run-time Data Mediator
Process Mediator
Choreography Engine
WSML Flight Reasoner

In addition to the set of packages hosted at SourceForge, the WSMX website<sup>1</sup> also makes nightly builds of the WSMX core and selected components available. A detailed list of all the improvements and changes of the WSMX prototype is available in the current CHANGELOG from the WSMX CVS<sup>2</sup>.

<sup>1</sup> See <http://www.wsmx.org>

<sup>2</sup> at SourceForge (<http://sourceforge.net/projects/wsmx>) under components/core/CHANGELOG

## 2.3 Installation Guidelines

### 2.3.1 WSMX Software Prerequisites

The following list specifies the needed third party products and required libraries to use the prototype:

- JDK 5.0 (Download from <http://java.sun.com/j2se/1.5.0/download.jsp>)

An external JavaSpaces implementation is no longer needed for this version of the prototype, as a local transport mechanism that acts like a virtual space has been added. This allows to run WSMX without a tuplespace as long as all components are deployed on a single instance on a single machine. Refer to the corresponding sections of the previous version of this deliverable [12] if a separate JavaSpaces implementation is preferred or needed (e.g. if WSMX components are to be distributed over different machines).

### 2.3.2 WSMX Installation Instructions

The WSMX microkernel may be configured via a properties file that is by convention named `config.properties` and located in the same directory as the kernel. This file is the kernel configuration and it is responsible for several configuration aspects of an instance of the WSMX microkernel, like the location of the `systemcodebase`, ports for the Web Console and SSH Console, or information on the used space.

The `systemcodebase` is a location on the filesystem where component implementations reside. This location is either discovered by WSMX itself or explicitly defined in the WSMX kernel configuration. Usually this is a directory populated with WSMX archives. A WSMX archive is a jar with a `.wsmx` extension and an agreed upon internal structure. The class files that make up the components implementations go to `/classes`, the archives deployment descriptor (if any) goes to `/META-INF`, libraries go jarred to `/lib`. WSMX uses custom classloaders that extract embedded jars, resolve load requests to the individual libraries, and provide isolation domains which allow components to load different version of the same class. WSMX scans the `systemcodebase` at startup and continues monitoring it to pick up components that are to be hot-deployed. Since everything that doesn't have a `.wsmx` extension is ignored, the WSMX executable jar, configuration, key and policy files are also found in the `systemcodebase`.

Components located in the `systemcodebase` will be individually and automatically scanned for a component configuration. Listing 1 shows a sample configuration properties file, as supplied with the current version of the prototype, after which a step-by-step instruction for setting up the WSMX server is given. The kernel configuration can now be specified in either Java Properties format or the previous XML format (as shown in Listing 1 of [12]).

```
#Set ports for the WSMX daemons and the space address
wsmx.spaceaddress=localhost
wsmx.httpport=8080
wsmx.sshport=8090
#The location of the systemcodebase can be set explicitly
#wsmx.systemcodebase=/home/wsmx/systemcodebase
```

**Listing 1: WSMX kernel configuration in Java Properties format**

- The WSMX core jar can be run from the command line with sufficient privileges granted. A sample policy file (policy.all) which grants unrestricted access is supplied with the release.

```
java -Djava.security.policy=/path/to/policy -jar WSMXcore.jar
```

- To deploy a component, package it in a WSMX archive and copy it to the systemcodebase. WSMX will discover it automatically and inject it into the running instance.
- You may monitor and administer WSMX through either the GUI-based HTTP management console or the TUI-based SSH management console. In addition the WSMX Management plug-in, integrated with WSMO Studio<sup>3</sup> [20], provides an User Interface for managing and interacting with the WSMX environment.

## 2.4 API information

The API for component interfaces (their sources, binaries and documentation) is available for download at: <http://sourceforge.net/projects/wsmx>. Third party component providers should download the newest version of the WSMX Integration API, in order for their components to be compatible with the WSMX architecture.

Additional APIs used for the architecture prototype include the WSMO API [2].

## 2.5 Licence information

WSMX uses the GNU General Public Licence<sup>4</sup>.

The third party software components and libraries included in the current WSMX release are using diverse licences, as shown in Appendix 2 of [12].

More detailed information about licensing of DIP components are provided in [5].

## 2.6 How to use the prototype

Following the steps detailed in section 2.3.2, WSMX should be running on a local machine. The WSMX Management Console can then be accessed by pointing your browser to <http://localhost:port>, where port is the port number which has been defined in the kernel configuration (the “config.properties” file) available with the WSMX packages.

The GUI documentation in section 3.1 details the possibilities of managing WSMX with the management console, including details on selecting and manipulating components.

## 2.7 Roadmap for future plans

The next version of the WSMX prototype (available in June 2006) will be a further refinement, also taking into account several new components, identified as necessary, but not yet developed (e.g. a security component). This version will also include the real

---

<sup>3</sup> available at <http://www.wsmostudio.org/>

<sup>4</sup> <http://www.opensource.org/licenses/gpl-license.php>

implementations of the remaining mock-up components, the Selector and Orchestration Engine.

### 3 ARCHITECTURE PROTOTYPE DOCUMENTATION

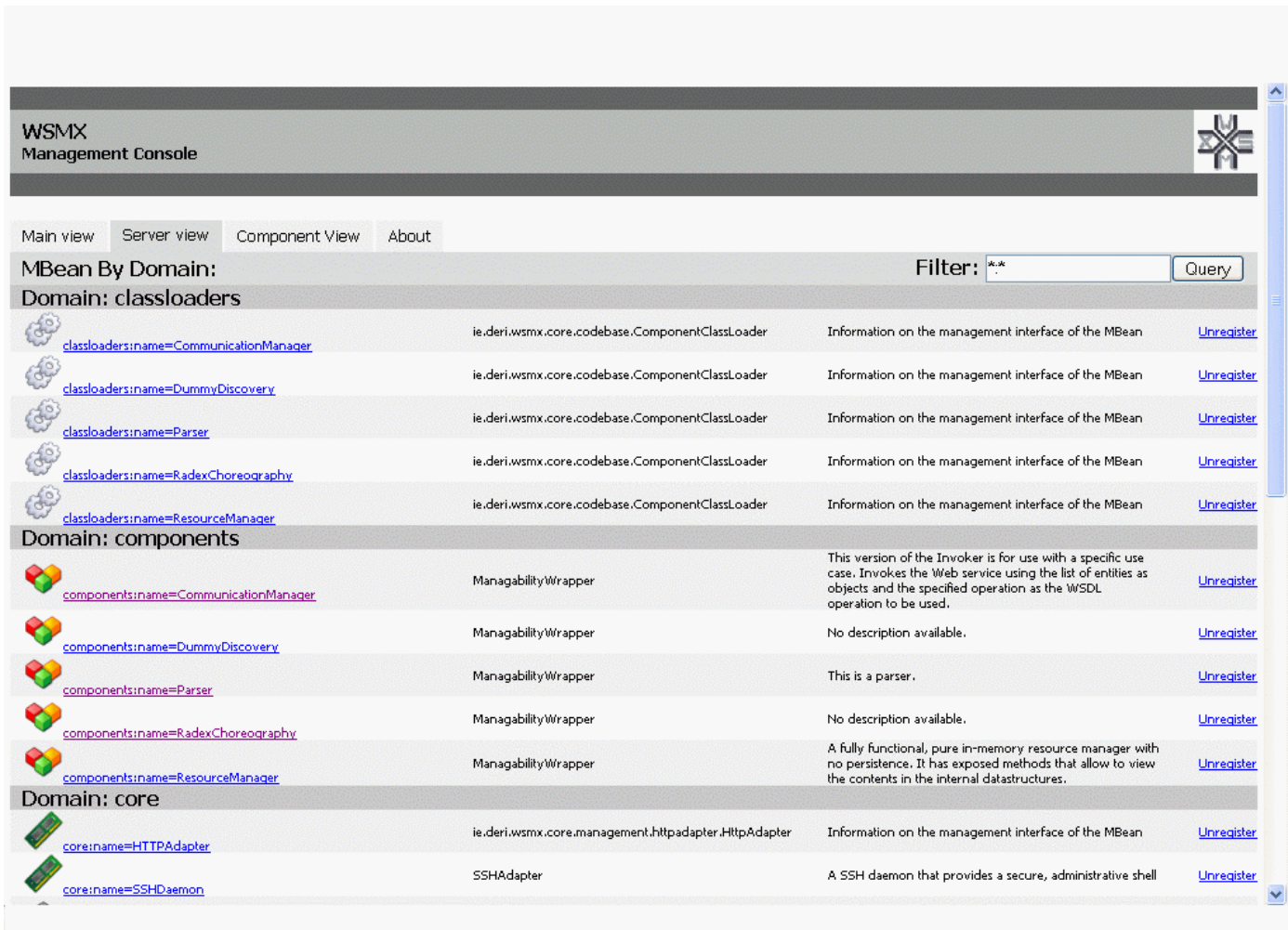
#### 3.1 Server Administration GUI

The WSMX Management Console facilitates basic server administration tasks. While the GUI has been updated since the last version of the prototype, the basic separation of available information in different views was retained. The views show general information on the running server instance (the main view, as seen in Figure 1), or allow for the administration of the deployed server components (the server view, as seen in Figure 2).



**Figure 1: WSMX Management Console – Main View**

As can be seen in Figure 2, the different services, deployed in the WSMX server as MBeans, are divided up by their domains. Besides the domain of actual WSMX components, several other important domains can be seen in the Server view, e.g. Classloaders, Core services and Loggers. Selecting one of the components from the server view opens up the MBean view of this service, where different service attributes can be manipulated and service methods can be executed.



The screenshot shows the WSMX Management Console interface. At the top, there is a header with the WSMX logo and the text 'WSMX Management Console'. Below the header, there are navigation tabs: 'Main view', 'Server view', 'Component View', and 'About'. The 'Server view' tab is selected. The main content area is titled 'MBean By Domain:' and includes a search filter box with the text '\*:\*' and a 'Query' button. The content is organized into three domains: 'classloaders', 'components', and 'core'. Each domain contains a list of MBeans with their names, class names, descriptions, and 'Unregister' links.

Domain	MBean Name	Class Name	Description	Action
Domain: classloaders	classloaders:name=CommunicationManager	ie.deri.wsmx.core.codebase.ComponentClassLoader	Information on the management interface of the MBean	Unregister
	classloaders:name=DummyDiscovery	ie.deri.wsmx.core.codebase.ComponentClassLoader	Information on the management interface of the MBean	Unregister
	classloaders:name=Parser	ie.deri.wsmx.core.codebase.ComponentClassLoader	Information on the management interface of the MBean	Unregister
	classloaders:name=RadexChoreography	ie.deri.wsmx.core.codebase.ComponentClassLoader	Information on the management interface of the MBean	Unregister
	classloaders:name=ResourceManager	ie.deri.wsmx.core.codebase.ComponentClassLoader	Information on the management interface of the MBean	Unregister
Domain: components	components:name=CommunicationManager	ManagabilityWrapper	This version of the Invoker is for use with a specific use case. Invokes the Web service using the list of entities as objects and the specified operation as the WSDL operation to be used.	Unregister
	components:name=DummyDiscovery	ManagabilityWrapper	No description available.	Unregister
	components:name=Parser	ManagabilityWrapper	This is a parser.	Unregister
	components:name=RadexChoreography	ManagabilityWrapper	No description available.	Unregister
	components:name=ResourceManager	ManagabilityWrapper	A fully functional, pure in-memory resource manager with no persistence. It has exposed methods that allow to view the contents in the internal datastructures.	Unregister
Domain: core	core:name=HTTPAdapter	ie.deri.wsmx.core.management.httpadapter.HttpAdapter	Information on the management interface of the MBean	Unregister
	core:name=SSHDaemon	SSHDaemon	A SSH daemon that provides a secure, administrative shell	Unregister

Figure 2: WSMX Management Console – Server View

## 3.2 WSMX Component descriptions

The following section provides short summaries of scope and functionalities for each of the currently available WSMX components.

### 3.2.1 Adapter Framework

Current Version: 0.1

Available at: WSMX CVS at SourceForge (<http://sourceforge.net/projects/wsmx/>) under the module /components/adapter/.

The data syntax adaptation is one of the known problems in application integration, where different applications, operating on different syntaxes to represent their data (e.g., XML, EDI, WSML, OWL, etc.), need to be interconnected. Semantic Web Services aim at facilitating application integration resolving data heterogeneity by using Ontologies. The Semantic Web Service platforms, however, can interact only with the applications that use the same syntax to represent their data. Therefore, to enable interoperability between applications using different data representations, data is transformed from one syntactic format to another using adapters in most cases. In other words, adapters transform data represented using the syntax used in the sending application to the syntax

used in the receiving application. On a pragmatic level, several of such adapters need to be developed for adapting syntaxes of applications used even within a single business organization. Thus to reduce the amount of effort needed for adapter development and to enable reuse of already developed adapters, an adapter framework is designed and implemented.

The adapter framework is implemented in Java. It serves as a common platform where application specific adapters can be deployed, undeployed, tested and used for adapting syntax used to represent data in one application to the syntax used to specify data in another application. It provides a skeleton adapter as a template for developing an application specific adapter. In other words, implementation of an application specific adapter should use the skeleton adapter provided by the adapter framework to indicate the entry point for accessing that particular adapter. The conceptual design of the adapter framework is applicable for any middleware platform that is used for the purpose of application integration. In the context of DIP, the adapter framework is implemented to suit the syntactic requirement of its execution platform.

The adapter framework is designed to support the deployment of a new adapter, undeploying the already deployed adapter, testing the functionality of the deployed adapter, reusing the deployed adapters, viewing available adapters and monitoring their usage. Its current implementation, however, serves as a Web service and supports only the first five features. Since the adapter framework is serving as a Web service, it is important that the deployed adapters are manipulated carefully. In the current version of the adapter framework no mechanism is yet implemented to ensure a secure manipulation of the deployed adapters. In the next version of the adapter framework a security mechanism will be implemented, such that only the owner of an adapter can undeploy it.

### **3.2.2 Communication Manager**

Current Version: 0.3

Available at: WSMX CVS at SourceForge (<http://sourceforge.net/projects/wsmx/>)

The communication manager is responsible for sending and receiving WSML messages to and from entities external to WSMX or adapters representing such external entities. In this context, the term entity means any external system acting as a service requester as well as services that are invoked by WSMX. Adapters are expected to be used where the external entity cannot directly support communication using WSML messages and translation to another data representation is required. Translation from WSML to another data representation is often referred to as “lowering” while translation from another data representation to WSML is often referred to as “lifting”.

#### **Sending Messages**

The communication manager provides the Invoker interface to allow other components in WSMX to make an invocation on a Web service. The Invoker interface requires the service description, grounding information (the endpoint to be invoked) and the data to be sent to the service.

Both the service description and data will be represented in WSML.

#### **Receiving Messages**

The communication manager provides an interface to external entities to accept WSML messages which may represent a goal to be achieved or be a message corresponding to a message exchange that has already been instantiated. The communication manager accepts the message and determines which execution semantics should be used to process the message further through WSMX.

### **Grounding to WSDL**

WSMX provides a reference implementation for using semantic service descriptions as a fundamental aspect of a service oriented architecture. However it must be possible for the DIP architecture to be compatible with WSDL, the current standard for describing Web services. To make this possible, the choreography section of the WSMO description describes how the in and out messages are grounded to WSDL operations. The choreography engine is responsible for sending this information to the Invoker interface of the communication manager when a service needs to be invoked.

### **3.2.3 Resource Manager**

Current Version: 0.1

Available at: WSMX CVS at SourceForge (<http://sourceforge.net/projects/wsmx/>)

The Resource Manager is responsible for persistent storage within WSMX. The component implementing this interface is responsible for storing every data item WSMX uses and that has persistency requirements. The WSMO API provides a set of Java interfaces that can be used to represent the domain model defined by WSMO. The WSMO4j<sup>5</sup> project includes both the WSMO API itself and its reference implementation. While it is not a pre-requisite that an implementation of the Resource Manager uses WSMO4j internally, all implementations must use the interface defined in the DIP API which is defined using WSMO4j classes.

Currently WSMX defines interfaces for six repositories. Four of these repositories correspond to the top level concept of WSMO i.e. Web services, ontologies, goals, and mediators. The fifth repository is used by WSMX for non-WSMO data items e.g. events and messages. Finally, the sixth repository is used to register WSDL documents and to link them to the corresponding WSMO descriptions. The WSDL descriptions are required to ground WSMO service descriptions to SOAP or SOAP/HTTP.

### **3.2.4 Run-time Data Mediator**

Current Version: 0.2

Available at: WSMX CVS at SourceForge (<http://sourceforge.net/projects/wsmx/>)

The WSMX Data Mediation Component has the role of supplying the reconciliation of the data heterogeneity problems that can appear during discovery, composition, selection or invocation of Web Services. This component is dependent on the ontology mappings created during design time and stored in a persistent storage in order to perform the transformation on the actual data in a completely automatic manner.

The design time tool that can be used to create these mappings is part of the Web Service Modeling Toolkit<sup>6</sup>. The component consists of a graphical interface that offers

---

<sup>5</sup> See <http://wsmo4j.sourceforge.net/>

<sup>6</sup> Available at <http://sourceforge.net/projects/wsmt>

support to the domain expert placing its inputs (i.e. mappings). The user can browse the ontology and can decide, based on the offered suggestion, which mappings would be more suited to capture the semantic similarities of the two ontologies. When the process is completed the mappings are stored in an external storage for further usage (by the run-time component or by the domain expert for further refinements).

The run-time component has the role to retrieve from the storage the already created mappings, to transform them into rules and finally to execute them against the incoming instances in order to obtain the target instances. Since the mappings represent the connection point between the two sub-components (design-time and run-time), one of the dependencies for the run-time component is on the mapping storage. This version of the mediator realises its independent mapping storage (a MySQL database, that can be populated by using the design time tool), but future plans include an extension of the storage interaction mechanism such as to be able to use the WSMX repository, for storing mappings.

Another crucial element of the runtime component is the reasoning system used for executing the rules in the final stage of the mediation process. For this the Flora2<sup>7</sup> engine together with its underlying system, XSB Prolog<sup>8</sup>, are used; they are integrated in Java by using InterProlog<sup>9</sup>, a Java front-end and enhancement for XSB Prolog. The installation of these systems is completely handled by a set of scripts coming with the mediation component, without any involvement from the user side.

The WSMX Data Mediator uses the WSMO4J v0.5 as object model for ontologies, and the Abstract Mapping Language API v0.1<sup>10</sup> for the mappings.

The functionality expected for the next iteration (December 2005) will contain additionally:

- **More flexible mechanism for mapping storage.** The current storage mechanism will be extended to support alternative ways for saving mappings (e.g. WSMX repository, file system)
- **Migrate to Abstract Mapping Language v0.2.** Migrate to a new version of the mapping API in order to ensure full interoperability with other DIP and non-DIP components.
- **Include more complex mapping rules.** More complex mismatches are to be addressed and the tools (both the design and the run-time) will be extended to generate and execute these mappings.

### 3.2.5 Process Mediator

Current Version: 0.1

Available at: WSMX CVS at SourceForge (<http://sourceforge.net/projects/wsmx/>)

---

<sup>7</sup> For more information and documentation check <http://flora.sourceforge.net/>

<sup>8</sup> For more information and documentation check <http://xsb.sourceforge.net/>

<sup>9</sup> For more information and documentation check <http://www.declarativa.com/interprolog/>

<sup>10</sup> See <http://www.omwg.org/>

The WSMX Process Mediator is one of the two Process Mediator prototypes developed as part of DIP deliverable 5.5 Business Data and Process-Level Mediation Module Prototype v2. It addresses the processes heterogeneity problem, which may occur during the communication between two partners (i.e., the requestor and the provider of a service). The processes considered are the public processes of the two participants in a conversation, expressing their behaviour as WSMO choreographies [10]. This version of the prototype exploits the results of the first mediation module prototype [8], by using the services of a data mediation module for transforming the instances sent by one of the partners in terms of its own ontology in instances expressed in terms of the targeted partner ontology.

The mediation of behaviours may be needed 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 our approach this adjustment happens in neither of the involved parties' sides, but in the Process Mediator [7]. 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.

The following types of mismatches can currently be addressed:

- a) **Stopping an unexpected message:** If one of the partners sends a message that the other one does not want to receive, the mediator should just retain and store it. This message (or a subset of the data contained in this message) can be sent later, if needed, or it can just be deleted after the communication ends.
- b) **Inverting the order of messages:** 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:** 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 in several different instances, which may appear during the data mediation process, is a subset of message splitting, since a message can consist of multiple instances.
- d) **Combining messages:** 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. Similarly with the previously described case, combining messages may also involve combining several instances in a single one during the data mediation process.

### 3.2.6 Choreography Engine

Current Version: v0.02b8

Available at: WSMX CVS at SourceForge (<http://sourceforge.net/projects/wsmx/>)

The choreography engine's responsibility is to support the behavioural aspect of the communication between the requester and provider. It takes the choreography descriptions of the involved parties and creates ontology state machine instances for each of them. Interactions between the two parties cause transition to fire in both of these machine instances, within the range of all possible interactions.

In compliance with the DIP/WSMX architecture the choreography engine tackles a single aspect of the general problem, which is orthogonal to all other aspects. Data- and Process Mediation happen transparently to the choreography engine between applying an updateset to one of the ontology ASM instances and starting the next step of the other ontology instance. The link to the Communication Manager, which carries out the actual invocation, manifests itself through an object representation of the grounding information that accompanies the respective transition rule that triggered the update, ultimately causing the invocation.

The prototype for December 2005 supports the latest specification of WSMO ontology ASMs, and is able to execute them with the help of a WSML reasoner. It is also fully integrated into the WSMX platform and provides the interaction hooks for the necessary components.

### 3.2.7 WSML Flight Reasoner

Current Version: 0.1

Available at: the CVS repository “[cvs.deri.at:/usr/local/cvsroot](http://cvs.deri.at:/usr/local/cvsroot)” (module `wsm2reasoner`)

The following section describes the internal version for December 2005 of the deliverable D1.9 WSML-Flight Reasoner [9]. The WSML-Flight reasoner is a base component within the WSMX architecture, for other components to perform reasoning on ontological descriptions expressed in the Flight-variant of the WSML ontology language. It is a *conforming* WSML-Flight reasoner, meaning that it handles all necessary constructs of this language variant according to the WSML semantics specification in deliverable D1.7 [19].

It is based on the KAON2 hybrid reasoning system as an underlying reasoning engine whose datalog functionality it uses to realise the rule-style inferencing that is characteristic for handling WSML-Flight ontologies. It is connected to the WSMO4J ontology management API which it uses to programmatically process the elements in WSML ontologies and logical expressions. The following figure depicts the basic functional principle by which it wraps the KAON2 functionality to make it available for reasoning in the WSML language.

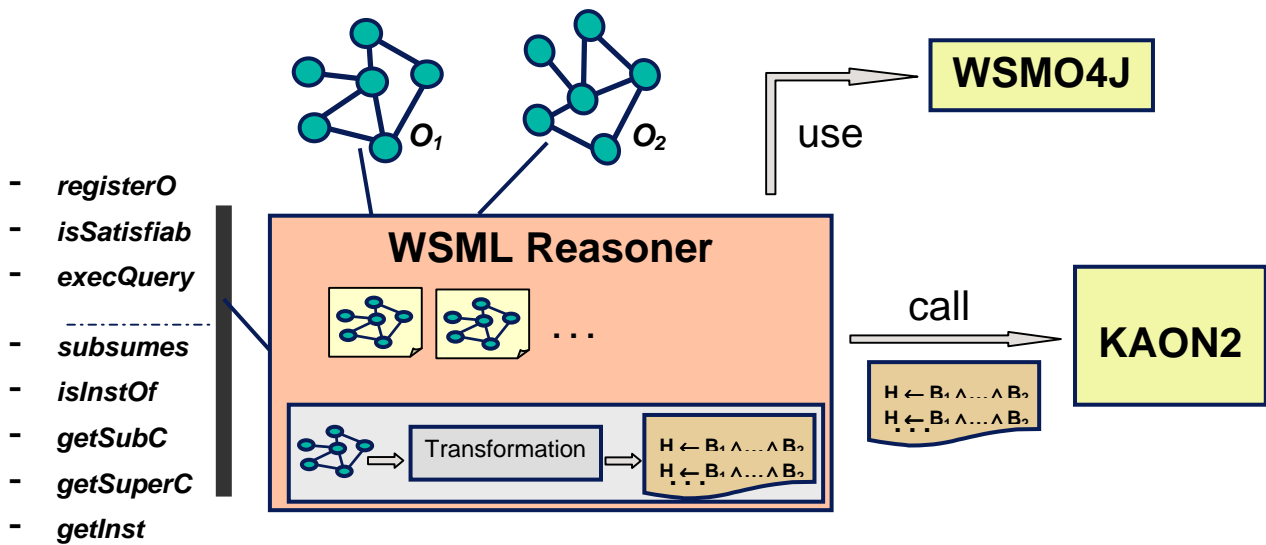


Figure 3: WSML Flight Reasoner

Ontologies to be reasoned with have to be registered to the component beforehand. They are then translated from WSML conceptual and logical expression syntax to datalog-style rules which can be processed by KAON2.

The WSML-Flight Reasoner offers the basic inference service of answering conjunctive queries, which is connected to the entailment of ground facts as specified in D1.7. Additionally, it offers convenience methods for checking subsumption of concepts in an explicit subsumption hierarchy and checking for instance relationship. In particular, the internal version for December 2005 covers the following features:

- retrieval: answering conjunctive queries (also some form of disjunction)
- consistency: checking ontologies for satisfiability (discover contradictions)
- WSML-Flight conceptual and logical expression syntax
- Conforming datatype reasoning (covering integer, decimal and string)

### 3.3 Additional documentation

The javadoc documentation for the WSMO API and its reference implementation `wsmo4j` can be found at <http://wsmo4j.sourceforge.net/reference.html>

In addition the current documentation for the WSMX Integration API is included in the relevant download on the WSMX project at SourceForge, while the basis for the Integration API can be found in the DIP deliverable on the component APIs [18], respectively. A detailed description of the current prototype architecture, the diverse entry points and the correlating execution semantics is provided in [11].

## 4 CONCLUSION

This deliverables provides an overview of the second version of the DIP architecture prototype including detailed installation guidelines and documentation. Nightly builds of the prototype can be downloaded from <http://www.wsmx.org/downloads.html>, while

the sources are available from SourceForge at <http://sourceforge.net/projects/wsmx>. For demonstration purposes a local installation following the installation guidelines described in this deliverable might be useful.

A possible use of the current version of the prototype, as well as the necessary backend services, and front-end GUI, can be seen in [6].

---

**REFERENCES**

- [1] Hauswirth, M. et al. (2004): *D6.2: DIP Architecture*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [2] Dimitrov, M.; Ognyanov, D.; Kiryakov, A. (2005): *D6.4: WSMO API*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [3] Kirov, V.; Kiryakov, A. (2005): *D6.3: DIP Component APIs*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [4] Zaremba, M. et al. (2005): *D6.5: DIP Revised architecture*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [5] De Saint Marie, C. (2005): *D13.2: Analysis of the appropriate open-source licensing schemata, including those used for related WS and B2B standards*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [6] Wahler, A. et al. (2005): *D8.4: Case Study implementation prototype v1.0*, DIP Project, WP8 Case Study B2B in Telecommunications. <http://dip.semanticweb.org>
- [7] Cimpian, E.; Lemcke, J.; Mocan, A.; Schumacher, M. (2005): *D5.3: Business Process-level Mediation Module Specification*, DIP Project, WP5 Service Mediation. <http://dip.semanticweb.org>
- [8] Drumm, C.; Domingue, J.; Cabral, L.; Mocan, A.; Corcho, O.; Atanassov, S. (2005): *D5.4: Business data and process-level mediation module prototype v1*, DIP Project, WP5 Service Mediation. <http://dip.semanticweb.org>
- [9] Motik, B.; Nagypal, G.; Grimm, S. (2005): *D1.9: WSMO Reasoner*, DIP Project, WP1 Ontology Reasoning and Querying. <http://dip.semanticweb.org>
- [10] Scicluna, J.; Polleres, A.; Roman, D.(eds) (2005): *Ontology-based Choreography and Orchestration of WSMO Services*, WSMO working draft, available at <http://www.wsmo.org/TR/d14/v0.2/>
- [11] Zaremba, M. et al. (2005): *D6.8: DIP Revised architecture v2.0*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [12] Haselwanter, T.; Schreder, B.; Wahler, A.; Zaremba, M. (2005): *D6.7: Architecture Prototype v1.0*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [13] Roman, D.; Lausen, H.; Keller, U. (2005): “*D2v1.2 Web Service Modelling Ontology (WSMO)*”, WSMO Working Draft 13 April 2005. <http://www.wsmo.org/TR/d2/v1.2/>
- [14] Cimpian, E.; Vitvar, T.; Zaremba, M. (2005): “*D13.0v0.2 Overview and Scope of WSMX*”, WSMX Working Draft 23 February 2005. <http://www.wsmo.org/TR/d13/d13.0/v0.2/>
- [15] Fensel, D.; Bussler, C.; Ding, Y.; Omelayenko, B. (2002a): *The Web Service Modeling Framework WSMF*. Electronic Commerce Research and Applications, 1(2), 2002.
- [16] Zaremba, M.; Oren, E. (2005): “*D13.2v0.2 WSMX Execution Semantics*”, WSMX Working Draft 14 July 2005. <http://www.wsmo.org/TR/d13/d13.2/v0.2/>
- [17] Vasiliu, L.; Moran, M.; Bussler, C.; Roman, D. (2004): “*D19.1v0.1 WSMO in DIP*”, WSMO Working Draft 21 June 2005. <http://www.wsmo.org/2004/d19/d19.1/v0.1/>
- [18] Kirov, V.; Kiryakov, A. (2005): *D6.9: DIP revised component APIs v2.0*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
- [19] De Bruijn, J.; Polleres, A.; Lausen, H.; Predoiu, L. (2005): *D1.7: Semantics*, DIP Project, WP1 Ontology Reasoning and Querying. <http://dip.semanticweb.org>
- [20] Simov, A.; Dimitrov, M.; Kerrigan, M.; Momtchev, V.; Hepp, M.; Henke, J.; Richardson, M. (2006): *D4.11: WSMO Studio v2*, DIP Project, WP 4b WSMO Platform & Tools. <http://dip.semanticweb.org>