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

Deliverable

WP 6: Interoperability and Architecture
D6.13
Software Deliverable and Installation Guidelines V3

Thomas Haselwanter
Bernhard Schreder
Alexander Wahler
Michal Zaremba

July 12th, 2006
SUMMARY

This deliverable provides an overview of the third version of the WSMX implementation prototype for the DIP architecture including detailed installation guidelines and documentation. The prototype itself consists of the WSMX core as well as several additional components. This document is an updated version of the deliverable describing the last prototype ([12]), and additionally contains descriptions of new components and updates to existing component descriptions from the previous version.

WSMX [14] is an execution environment which enables discovery, selection, mediation, invocation and interoperation of 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 means of achieving dynamic interoperability between 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

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

Deliverable Number 6.13 Title Software Deliverable and Installation Guidelines V3

Work package Number 6 Title Interoperability and Architecture

Date of delivery Contractual M30 Actual 12-Jul-06

Status version. 1.0 final

Nature Prototype ✔ Report ☐ Dissemination ☐ Ontology ☐

Dissemination Level Public ✔ Consortium ☐

Authors (Partner) Thomas Haselwanter (UIBK), Bernhard Schreder (Hanival), Alexander Wahler (Hanival), Michal Zaremba (UIBK)

Responsible Author Bernhard Schreder Email bernhard.schreder@hanival.net

Partner Hanival Phone +4313195843-13

Abstract (for dissemination) This deliverables provides an overview of the third version of the WSMX implementation prototype for the DIP architecture.

Keywords Execution environment, architecture, semantic web, semantic web services, execution semantics, prototype

Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>22-may-06</td>
<td>001</td>
<td>Bernhard Schreder</td>
<td>Initial version</td>
</tr>
<tr>
<td>19-jun-06</td>
<td>002</td>
<td>Bernhard Schreder</td>
<td>Integrated component descriptions</td>
</tr>
<tr>
<td>03-jul-06</td>
<td>003</td>
<td>Bernhard Schreder</td>
<td>Final version sent to reviewers</td>
</tr>
<tr>
<td>11-jul-06</td>
<td>004</td>
<td>Bernhard Schreder</td>
<td>Integrated reviewers’ comments</td>
</tr>
<tr>
<td>Reviewer</td>
<td>Email</td>
<td>Phone</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Barry Norton</td>
<td><a href="mailto:b.j.norton@open.ac.uk">b.j.norton@open.ac.uk</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mick Kerrigan</td>
<td><a href="mailto:mick.kerrigan@deri.org">mick.kerrigan@deri.org</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partner OU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partner NUIG</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Ireland Galway | NUIG | Dr. Sigurd Harand  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Ireland  
Email: sigurd.harand@deri.org  
Tel: +353 91 495112 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. BankInter  
Paseo Castellana, 29  
28046 Madrid, Spain  
Email: mmitner@bankinter.es  
Tel: 916234238 |
| British Telecommunications Plc. | BT | Dr John Davies  
BT Exact (Orion Floor 5 pp12)  
Aldastra Park Martlesham  
Ipswich IPS 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 |
| 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 |
<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| 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 |
| MDR Partners | MDR | Rob Davies  
MDR Partners  
8 St. Andrew Street  
Hertford, Herts.  
United Kingdom, SG14 1JA,  
Email: rob.davies@mdrpartners.com  
+44 (0)208 8763121 |
| Hanival Internet Services GmbH | HANIVAL | Alexander Wahler  
Hanival Internet Services GmbH  
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: jbdomingue@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>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sirma AI Ltd.</td>
<td>Sirma</td>
<td>Atanas Kiryakov, Ontotext Lab, 3rd Floor Office Express IT Centre, 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria. Email: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a> Tel.: +359 2 9768 303</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Unicorn</td>
<td>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</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>VUB</td>
<td>Pieter De Leenheer, Starlab- VUB, Vrije Universiteit Brussel, Pleinlaan 2, G-10, 1050 Brussels, Belgium. Email: <a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a> Tel.: +32 (0) 2 629 3749</td>
</tr>
</tbody>
</table>
**LIST OF KEY WORDS/ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>Abstract State Machine</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>SEE</td>
<td>Semantic Execution Environments</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>TUI</td>
<td>Text User Interface</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>WSMT</td>
<td>Web Service Modeling Toolkit</td>
</tr>
<tr>
<td>WSMX</td>
<td>Web Service Execution Environment</td>
</tr>
</tbody>
</table>
# Table of Contents

**Summary** .......................................................................................................................... I

**List of Key Words/Abbreviations** ..................................................................................... VII

**Table of Contents** ............................................................................................................ VIII

1 **Introduction** .................................................................................................................... 1

2 **Architecture Prototype Fact Sheet** .................................................................................. 1
   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 ................................................................................................................ 5
   2.5 Licence information .......................................................................................................... 5
   2.6 How to use the prototype ................................................................................................ 5
   2.7 Roadmap for future plans ................................................................................................ 5

3 **Architecture Prototype Documentation** .......................................................................... 6
   3.1 Server Administration GUI ............................................................................................... 6
   3.2 Server Administration TUI ................................................................................................ 7
   3.3 WSMX Component descriptions .......................................................................................... 8
       3.3.1 Adapter Framework ..................................................................................................... 8
       3.3.2 Communication Manager .......................................................................................... 9
       3.3.3 Discovery Engine ........................................................................................................ 9
       3.3.4 Resource Manager ...................................................................................................... 10
       3.3.5 Choreography Engine ............................................................................................... 10
       3.3.6 Orchestration Engine ................................................................................................. 11
       3.3.7 WSML Flight Reasoner ............................................................................................... 11
   3.4 Additional documentation ................................................................................................. 13

4 **Conclusion** .................................................................................................................... 13

**References** ........................................................................................................................ 14

**List of Figures**

Figure 1: WSMX Management Console – Main View ............................................................... 6
Figure 2: WSMX Management Console – Server View .............................................................. 7
Figure 3: WSMX SSH console................................................................. 7
Figure 4: 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 third version of the WSMX implementation prototype for the DIP architecture, representing an implementation of the architecture first specified in [1], revised in [4] and [11] and last updated in [21]. The prototype itself consists of the updated WSMX core server 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 WSMX implementation 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 first version of the 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 and tutorials.

2.2 Available Release and Components

The current release of WSMX (version 0.3) 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

<table>
<thead>
<tr>
<th>Package name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsmx_components_bin-0.3</td>
<td>Compiled versions of available WSMX components (packaged as WSMX archives)</td>
</tr>
<tr>
<td>wsmx_components_src-0.3</td>
<td>Sources for all components</td>
</tr>
<tr>
<td>wsmx_core_bin-0.3</td>
<td>Compiled version of core engine and mock-up components</td>
</tr>
<tr>
<td>wsmx_core_src-0.3</td>
<td>Sources without third party libraries</td>
</tr>
<tr>
<td>wsmx-integration-api-0.3</td>
<td>Contains the SEE Integration API and corresponding javadoc</td>
</tr>
</tbody>
</table>

The core packages include a number of mock-up components as described in [12]; in addition to the mock-up components the currently implemented components are available with the components package. Table 2 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 the various component deliverables in work packages 1 to 5.

Table 2: Real WSMX components

<table>
<thead>
<tr>
<th>Component name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter Framework</td>
</tr>
<tr>
<td>Communication Manager</td>
</tr>
<tr>
<td>Discovery Engine</td>
</tr>
<tr>
<td>Resource Manager</td>
</tr>
<tr>
<td>Run-time Data Mediator</td>
</tr>
<tr>
<td>Orchestration Engine</td>
</tr>
<tr>
<td>Process Mediator</td>
</tr>
<tr>
<td>Choreography Engine</td>
</tr>
<tr>
<td>WSML Flight Reasoner</td>
</tr>
</tbody>
</table>

In addition to the set of packages hosted at SourceForge, the WSMX website\(^1\) also offers nightly builds of the WSMX core and selected components. A detailed list of all

---

\(^1\) See http://www.wsmx.org
the improvements and changes of the WSMX prototype is available in the current CHANGELOG from the WSMX CVS.

2.3 Installation Guidelines

2.3.1 WSMX Software Prerequisites

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

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

An earlier version of this prototype also required an external JavaSpaces implementation. This is no longer a necessity for this version, as a local transport mechanism that acts like a virtual space has been added. This allows WSMX to be run 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 going 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 is responsible for several configuration aspects of an instance of the WSMX microkernel, for example 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 file system 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 archive’s 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 start-up 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.

at SourceForge (http://sourceforge.net/projects/wsmx) one can browse the CVS and view the changelog under components/core/CHANGELOG

---

2 at SourceForge (http://sourceforge.net/projects/wsmx) one can browse the CVS and view the changelog under components/core/CHANGELOG
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.

```java
#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

In the following section a step-by-step instruction for setting up the WSMX server is given:

- Download a WSMX binary distribution from SourceForge or get the latest nightly build from http://www.wsmx.org/downloads.html.

- Optional step: Start up a JavaSpaces compliant space implementation such as Outrigger or Blitz. If WSMX does not find a space during boot time it will substitute a virtual space that works as long as components only have local communication requirements.

- The microkernel within the executable wsmx.core can be run from the command line, given that sufficient privileges are granted:

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

  (A sample policy file (policy.all) which grants unrestricted access is supplied with the release.)

- To deploy a developed component, copy the packaged component archive 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 web console or the TUI-based SSH console. Point your browser to http://localhost:port, where port is the port number, which has been defined in the kernel configuration. The default port is 8080, if left undefined or as a fallback for invalid ports. Point your SSH client to localhost and login with user root at the port defined in the configuration. In addition the WSMX Management plug-in, integrated with the Eclipse-based WSMT\(^3\), provides a User Interface for managing and interacting with the WSMX environment.

- You may shutdown WSMX through any of its management consoles or by pressing Ctrl-C on the Operating System console (Ctrl-C does not work from within the eclipse console as it is not bound to an Operating System process kill signal in this 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. All implementations of the DIP architecture, such as WSMX or IRS, conform to the Semantic Execution Environment (SEE) Integration API. Third party component providers should download the newest version of the SEE Integration API, in order for their components to be compatible with the DIP architecture.

A WSMX component is a set of libraries, configurations and java classes, one of which implements an Interface from the DIP Architecture, all assembled in an archive, as explained in section 2.3.2. Even though it is preferred, for setup simplicity reasons, it is not necessary to have the whole component included in the archive. One may choose to develop a component as a web service, and have the wsmx archive act as a stub for example.

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

2.5 Licence information

WSMX uses the GNU Lesser General Public Licence$^4$. The third party software components and libraries included in the current WSMX release are using diverse licences, as shown in Appendix 2 of [23].

More detailed information about licensing of DIP components is 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 at the end of the DIP project in December 2006) will feature some final refinements, and add the last missing components to its collection of available ones. This version will also include the real implementations of the last remaining mock-up components, e.g. the Selector.

Further development (and integration with other technologies) of WSMX is ongoing in other projects, e.g. the SUPER project$^5$, another FP6 Integrated Project, which attempts to integrate Semantic Web Services technology with the Business Process Management area.

---

$^4$ http://www.opensource.org/licenses/lgpl-license.php

$^5$ http://www.ip-super.org/
3 ARCHITECTURE PROTOTYPE DOCUMENTATION

3.1 Server Administration GUI

The microkernel hosts an HTTP daemon that provides a GUI console for administrative tasks. This 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).

![WSMX Management Console](image)

**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 the 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.

---

6 an MBean (managed bean) is a Java object that represents a manageable resource, such as an application, a service, a component, or a device
Figure 2: WSMX Management Console – Server View

3.2 Server Administration TUI

Besides the GUI described above, the microkernel hosts an SSH daemon that provides a TUI console for administrative tasks. Figure 3 shows a screenshot of the TUI, on which the basic commands available can be seen.

Figure 3: WSMX SSH console
3.3 WSMX Component descriptions

The following section provides short summaries of scope and functionalities for each of the currently available WSMX components. A number of components (the Run-time Data Mediator and Process Mediator) have had no significant changes since the last version of this deliverable. Their descriptions can be found in [12].

3.3.1 Adapter Framework

Current Version: 0.2

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

The data syntax adaptation is one of the well 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. In most cases, to enable interoperability between applications using different data representations, data is transformed from one syntactic format to another using adapters. 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 has been designed and developed.

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 imposes an automated adaptation technique based on design-time conceptual mappings between the meta-models used to represent data by each communicating application. These mappings are translated to executable rules at runtime. Also, the conceptual design 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. There are two aspects to be considered regarding adapter security. On one hand, the adapter framework must check the structural integrity of the adapter before deploying it and the validity of the owner before undeploying it. On the other hand, the run-time validation at the level of message transfer must be performed. In the current version of the adapter framework no
mechanism is yet implemented to ensure a secure manipulation of the messages being transferred through the adapter but the structural integrity and owner validity are checked. Regarding performance, the adapter framework provides load monitoring of the deployed adapters. In case of overloading, new adapter instances are brought to execution mode from a pool of idle instances.

### 3.3.2 Communication Manager

**Current Version:** 0.3 - communicationmanager-200606.wsmx  
**Available at:** [http://www.wsmx.org/downloads.html](http://www.wsmx.org/downloads.html)

The CommunicationManager is responsible for dealing with the protocols for sending and receiving messages to and from WSMX. Its behaviour is accessed through the required Invoker, Receiver, EntryPoint and WSMORegistry interfaces of the SEE Integration API.

The Invoker interface is used by other functional components of the WSMX prototype to invoke operations on application Web services or on services representing the Goal a service requester wishes to achieve. For example, in a B2B scenario such as that described by the RosettaNet PIP3A4 process, the action of the service requester sending a purchase order message and receiving a purchase order acknowledgement (POA) message may be asynchronous. In such a case, WSMX needs to be able to invoke the service requester (sending the PO Goal) to send back the POA. In terms of the `achieveGoal` execution semantics, service-invocation is triggered by the choreography engine when a specific transition rules for a choreography instance is fired.

The Receiver interface acts as the entry point for messages being sent for a message exchange that has already been established between a service requester and provider via WSMX. Messages are sent either by the service requester or provider and include the data content of the message as well as a context token that identifies the conversation for which the message is intended. The context will have been provided by WSMX when the conversation was initiated.

The EntryPoint interface is used to start one of the defined execution semantics for WSMX. In this release, only the `achieveGoal` execution semantics is implemented.

The WSMORegistry interface is intended to allow WSMX be used as a repository of WSMO descriptions. In this release of the prototype, the 'store' method of this interface is implemented, which serves as a common entry point allowing WSML descriptions to be saved to the WSMX repository.

### 3.3.3 Discovery Engine

**Current Version:** 0.1  
**Available at:** WSMX CVS at SourceForge ([http://sourceforge.net/projects/wsmx/](http://sourceforge.net/projects/wsmx/))

The WSMX Discovery Component has the role of matching formalized goals with formalized Web service descriptions, and selecting the Web services that are relevant for the request.

The component is organized as a framework that matches the goal with the available Web service descriptions in two sequential steps. In the first optional step is a non-semantic pre-filtering that reduces the set of Web services. The technique used in this step is keyword-based matching, the filtering parameters being set such as to ensure that
no potentially matching Web services are filtered out. The second step corresponds to discovery based on simple descriptions of services (also known as lightweight semantic discovery). The resulting set of matching Web services then returned to the framework. Both keyword and lightweight discovery engines are WSMX standalone components, and can also be used separately. They are, as well as the framework, fully integrated into the WSMX platform and provide the interaction hooks for the necessary components.

The lightweight discovery engine models Web services and goals as sets of objects. More precisely, a Web service is represented as a set of objects that it delivers in its domain of value, while a goal is represented as a set of elements, which are relevant to the client as the outcome of a service execution. In a state-based description approach both the pre-state and the post-state with respect to service execution would be modelled explicitly. The sets of objects are modelled through concepts and relations described in ontologies that capture general knowledge about the problem domain under consideration.

The semantic matchmaking process decides whether a Web service offer is relevant for the request by analyzing the semantic descriptions of both. Since such descriptions are expressed in terms of the WSMO ontological model using the WSML knowledge representation language, semantic matchmaking is based on automated reasoning techniques, achieved with the help of a WSML reasoner.

### 3.3.4 Resource Manager

Current Version: 0.3 - resourcemanager-200606.wsmx
Available at: http://www.wsmx.org/downloads.html

The Resource Manager is responsible for providing a persistence layer for WSMX both for the WSMO descriptions used by WSMX and the events and states used by the Core component of WSMX to manage its operational state. The internal representations of the WSMO descriptions are provided in terms of the WSMO4J Java class model. The current version of the resource manager uses an in-memory database. This means that all descriptions stored by the Resource Manager are lost once the running instance of WSMX has been stopped. To facilitate easier loading of descriptions of ontologies, web service, goals and mediators, these can be placed in a "resources\resourcemanager" folder and will be automatically picked up by WSMX and stored by the Resource Manager.

Currently WSMX defines interfaces for five 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 used by the Core component to manage the operation of WSMX.

### 3.3.5 Choreography Engine

Current Version: v0.03b20
Available at: WSMX CVS at SourceForge (http://sourceforge.net/projects/wsmx/)

---

7 Available at http://wsmo4j.sourceforge.net/
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 transitions to fire in both of these machine instances, within the range of all possible interactions.

In compliance with the DIP 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 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.

### 3.3.6 Orchestration Engine

**Current Version:** B300606

Available at: [http://www.wsmx.org/downloads.html](http://www.wsmx.org/downloads.html)

The scope of this version of the DIP orchestration component is to execute orchestration interfaces of composite services. From a conceptual point of view, a service orchestration describes how its functionality is implemented by orchestrating other services functionalities. In WSMO/WSMX a service orchestration interface is implemented as an Ontologized Abstract State Machine (OASM).

This component registers first a service orchestration interface after parsing its WSML file description. Then, given a (input) sent message, it navigates through the OASM, invoking when indicated/necessary the component services functionalities, according to the process execution logic.

The deployment package of this version of the orchestration component, as well as additional documentation is available on the DIP BSCW server at Deliverables M30 / D4.20. The deliverable 4.20 also includes a comprehensive demonstration of the functionality of the prototype through an e-banking use case scenario. The demonstration shows how the orchestration component registers the orchestration interface from the service description after parsing its WSML file description. It illustrates also the states change of the OASM (after eventually invoking other services functionalities). An illustrated movie demonstrates the main functionalities of this component.

### 3.3.7 WSML Flight Reasoner

**Current Version:** 1.0

Available at: the CVS repository “cvs.deri.at:/usr/local/cvsroot” (module wsml2reasoner)

The WSML-Flight reasoner delivered with D1.9 [9] 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. In its final version 1.0, it is a conforming WSML-Flight reasoner, meaning that it handles all constructs of this language variant according to the WSML semantics specification in [19] and in [22].

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

![Figure 4: WSML Flight Reasoner](image)

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 services of checking an ontology for consistency and of answering conjunctive queries, which is connected to the entailment of ground facts. Additionally, it offers convenience methods for checking subsumption of concepts in an explicit subsumption hierarchy and checking for instance relationship. It also provides debugging functionality for reporting violated constraints.

The following features are supported by the system:

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

The component is maintained in the CVS repository “cvs.deri.at:/usr/local/cvsroot” (module wsml2reasoner) hosted by UIBK. The deployment package of the final version of the WSML-Flight reasoner is also available on the DIP BSCW server at WP1/D1.9/DIP_D1.9_deployment.zip.

3.4 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 SEE 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 deliverable provides an overview of the third version of the WSMX implementation prototype for the DIP architecture, 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 can be seen as part of the e-banking demonstrator in deliverable D4.20 (available on the DIP BSCW server at Deliverables M30 / D4.20).
REFERENCES


