



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

D6.17

Architecture Prototype V4

Matthew Moran

Bernhard Schreder

Thomas Haselwanter

Maciej Zaremba

Brahmananda Sapkota

December 31st, 2006



SUMMARY

This deliverable provides an overview of the fourth 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 full-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 the open source implementation of the system, which is used as a reference implementation of the DIP architecture is reported.

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

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

Deliverable	Number	6.17	Title	Architecture Prototype V4
Work package	Number	6	Title	Interoperability and Architecture

Date of delivery	Contractual	M36	Actual	31-Dec-06
Status	version. 1.0		final <input checked="" type="checkbox"/>	
Nature	Prototype <input checked="" type="checkbox"/> Report <input type="checkbox"/> Dissemination <input type="checkbox"/> Ontology <input type="checkbox"/>			
Dissemination Level	Public <input checked="" type="checkbox"/> Consortium <input type="checkbox"/>			

Authors (Partner)	Matthew Moran (NUIG), Bernhard Schreder (Hanival), Thomas Haselwanter (UIBK), Maciej Zaremba (NUIG), Brahmananda Sapkota (NUIG), Alexander Wahler (Hanival)			
Responsible Author	Matthew Moran		Email	Matthew.moran@deri.org
	Partner	NUIG	Phone	+ 353 91 495017



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			
Issue Date	Rev No.	Author	Change
22-may-06	001	Bernhard Schreder	Initial version
19-jun-06	002	Bernhard Schreder	Integrated component descriptions
03-jul-06	003	Bernhard Schreder	Final version sent to reviewers
11-jul-06	004	Bernhard Schreder	Integrated reviewers' comments
Nov 2006	005	Brahmananda Sapkota,	Update chapter 2, chapter 3 (to reflect new discovery and QoS discovery implementation, update to




		Matthew Moran, Maciej Zaremba, Thomas Haselwanter	reasoner from M30). Update chapter 4 to bring up to current status of WSMX. Overall modifications to reflect the final status of the DIP architecture prototype.
31 Dec 2006	1.0	Matthew Moran	Update following review comments.

Reviewer			
	Pieter De Leenheer	Email	pieter.de.leenheer@vub.ac.be
	Partner Vrije Universiteit Brussel	Phone	
	Michael Kerrigan	Email	Michael.stollberg@deri.org
	Partner University of Innsbruck	Phone	

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: sigurd.harand@deri.org Tel: +353 91 495112
Fundacion De La Innovacion.Bankinter	Bankinter  ebankinter.com	Monica Martinez Montes Fundacion de la Innovacion. Bankinter Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
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  Essex County Council	Mary Rowlett, Essex County Council PO Box 11, County Hall, Duke Street Chelmsford, Essex, CM1 1LX United Kingdom. Email: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI 	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany Email: abecker@fzi.de Tel: +49 721 9654 0
Institut für Informatik, Leopold-Franzens Universität Innsbruck	UIBK  universität innsbruck	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

Partner	Acronym	Contact
ILOG SA	<p>ILOG</p>  <p>Changing the rules of business</p>	<p>Christian de Sainte Marie 9 Rue de Verdun, 94253 Gentilly, France Email: csma@ilog.fr Tel: +33 1 49082981</p>
inubit AG	<p>Inubit</p> 	<p>Torsten Schmale inubit AG Lützowstraße 105-106 D-10785 Berlin Germany Email: ts@inubit.com Tel: +49 30726112 0</p>
Intelligent Software Components, S.A.	<p>iSOCO</p> 	<p>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</p>
MDR Partners	<p>MDR</p> 	<p>Rob Davies MDR Partners 8 St. Andrew Street Hertford, Herts. United Kingdom, SG14 1JA, Email: rob.davies@mdrpartners.com +44 (0)208 8763121</p>
Hanival Internet Services GmbH	<p>HANIVAL</p> 	<p>Alexander Wahler Hanival Internet Services GmbH Kirchengasse 13/1a A-1070 Wien Email: wahler@niwa.at Tel: +43(0)1 3195843-11 </p>
The Open University	<p>OU</p>  <p>The Open University</p>	<p>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</p>
SAP AG	<p>SAP</p> 	<p>Dr. Elmar Dörner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany Email: elmar.dorner@sap.com Tel: +49 721 6902 31</p>

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

LIST OF KEY WORDS/ABBREVIATIONS

ASM	Abstract State Machine
GUI	Graphical User Interface
IDE	Integrated Development Environment
SEE	Semantic Execution Environments
SOA	Service Oriented Architecture
SSH	Secure Shell
SWS	Semantic Web Services
TUI	Text User Interface
WSMO	Web Service Modeling Ontology
WSMT	Web Service Modeling Toolkit
WSMX	Web Service Execution Environment

TABLE OF CONTENTS

SUMMARY.....	II
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	6
2.6 How to use the prototype.....	6
2.6.1 Invoking WSMX through WSMT.....	6
2.6.2 Invoking WSMX as a Web Service.....	8
2.6.3 Examples of WSML Resources	8
2.7 Roadmap for future plans	8
3 ARCHITECTURE PROTOTYPE DOCUMENTATION.....	8
3.1 Server Administration GUI.....	8
3.2 Server Administration TUI.....	9
3.3 WSMX Component descriptions	10
3.3.1 Discovery Engine	10
3.3.2 Orchestration Engine	11
3.3.3 WSML Flight Reasoner.....	11
3.4 Additional documentation	11
4 RELATION TO WSMO4J AND WSMO STUDIO.....	12
5 CONCLUSION	12
REFERENCES	13

LIST OF FIGURES

Figure 1: WSMT SEE Perspective	7
Figure 2: WSMX Management Console – Main View.....	8

Figure 3: WSMX Management Console – Server View..... 9
Figure 4: WSMX SSH console..... 10
Figure 5: WSMX Relationship to Other DIP Products 12

LIST OF TABLES

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

1 INTRODUCTION

This deliverable provides the documentation for the fourth version of the WSMX implementation prototype for the DIP architecture, representing an implementation of the architecture first specified in [1], revised in [4], [11], [21], [23] and last updated in [23]. 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 Maciej Zaremba (maciej.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 0.4) 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_components_bin-0.4	Compiled versions of available WSMX components (packaged as WSMX archives)
wsmx_components_src-0.4	Sources for all components
wsmx_core_bin-0.4	Compiled version of core engine and mock-up components
wsmx_core_src-0.4	Sources without third party libraries
wsmx-integration-api-0.4	Contains the SEE Integration API and corresponding javadoc

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

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

In addition to the set of packages hosted at SourceForge, the WSMX website¹ 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 CVS².

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>)
- If QoS discovery is required (optional) then the Apache Derby database must be installed and initialised to support this component's implementation. The recommended installation is to deploy Derby as a `.war` archive file to an Apache Tomcat Web server installation. the Derby database can be started by going through the Tomcat manager application. To initialize a QoS database with sample QoS data, follow the instructions at http://wiki.wsmx.org/index.php?title=QoS_Component_Sample_Data

Note: 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.

Error! Reference source not found. shows a sample configuration properties file, as supplied with the current version of the prototype.:

```
#Set ports for the WSMX daemons and the space address
wsmx.spaceaddress=localhost
wsmx.httpport=8080
wsmx.axisport=8050
wsmx.sshport=8090

#The location of the systemcodebase can be set explicitly
#wsmx.systemcodebase=/home/wsmx/systemcodebase

#location of resources
```

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

```
wsmx.resourcemanager.ontologies = "${resources}/Ontologies"
wsmx.resourcemanager.goals = "${resources}/Goals"
wsmx.resourcemanager.webservices = "${resources}/WebServices"

#use the QoS discovery (or not)
wsmx.discovery.qosdiscovery = true
wsmx.discovery.qosdiscovery.createDB = false

#various log4j logging settings
log4j.appender.C=org.apache.log4j.ConsoleAppender
#log4j.appender.C.layout=org.apache.log4j.PatternLayout
...
```

Listing 1: WSMX kernel configuration in config.properties

The following settings are possible in `config.properties`:

- Assignment of ports for http, axis and ssh access.
- Explicit setting of the directory where the WSMX system codebase can be found
- Setting of resource directories from which WSMX will load WSMML ontologies, goals and Web services
- Setting to use the QoS discovery as part of overall discovery or not. (Note: if this is used, an Apache database needs to be installed and initialised with QoS data.)
- Various Log4j settings for logging, refer to the log4j documentation for specific information.
- It's also possible to configure the SSH username and password for the SSH interface.

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 internal structure defined for WSMX. 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 (deployed after WSMX has started). Since everything that doesn't have a `.wsmx` extension is ignored by the loading mechanisms, 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.

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

- Download the WSMX v0.4 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 Outtrigger 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 the necessary system access is supplied with the release. See the Java documentation on the `java.security.policy` and this [serverside.com](#) article³ class for more details)

- 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.
- WSMX can be monitored or administered either through the GUI-based web console or the TUI-based (TUI = Terminal User Interface) 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. (Note: if port 8080 is already taken e.g. by a Tomcat installation, WSMX will attempt to take port 8081 and so on.) Point the SSH client to localhost and login with root as a user name at the port defined in the configuration. In addition, the WSMX Management plug-in, integrated with the Eclipse-based WSM⁴, provides a User Interface for managing and interacting with the WSMX environment.
- To shutdown WSMX, use any of its management consoles or press 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://wiki.wsmx.org/index.php?title=WSMX_API. All implementations of the DIP architecture, such as WSMX or IRS, conform to the

³ http://www.theserverside.com/tt/articles/article.tss?l=dm_javaPolicy

⁴ Web Service Modeling Toolkit: a collection of tools for WSMO, WSML and WSMX, available at <http://sourceforge.net/projects/wsmx>

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.

Each 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. It is preferred, for setup simplicity reasons, but not necessary to have a component fully 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 WSMO4j API⁵ [2].

2.5 Licence information

WSMX uses the GNU General Public Licence⁶.

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 a Web 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.

2.6.1 Invoking WSMX through WSMT

- Download WSMT from <http://sourceforge.net/projects/wsmt>.
- Click on the SEE tab
- Right click the mouse anywhere in the left pane.
- Select New→WSMX Server
- Fill in the values for Host (e.g. localhost), HTTP Port (e.g. 8081) and Axis Port (e.g. 8050)
- Click Ok
- WSMT will connect to the specified WSMX server and will locate available Web service, ontology, mediator and goal descriptions.

⁵ <http://wsmo4j.sourceforge.net/>

⁶ <http://www.opensource.org/licenses/gpl-license.php>

- Using the SOAP Message View in the right-hand pane, Goals can be sent the achieveGoal entry point of WSMX. The outgoing and incoming SOAP messages are displayed.

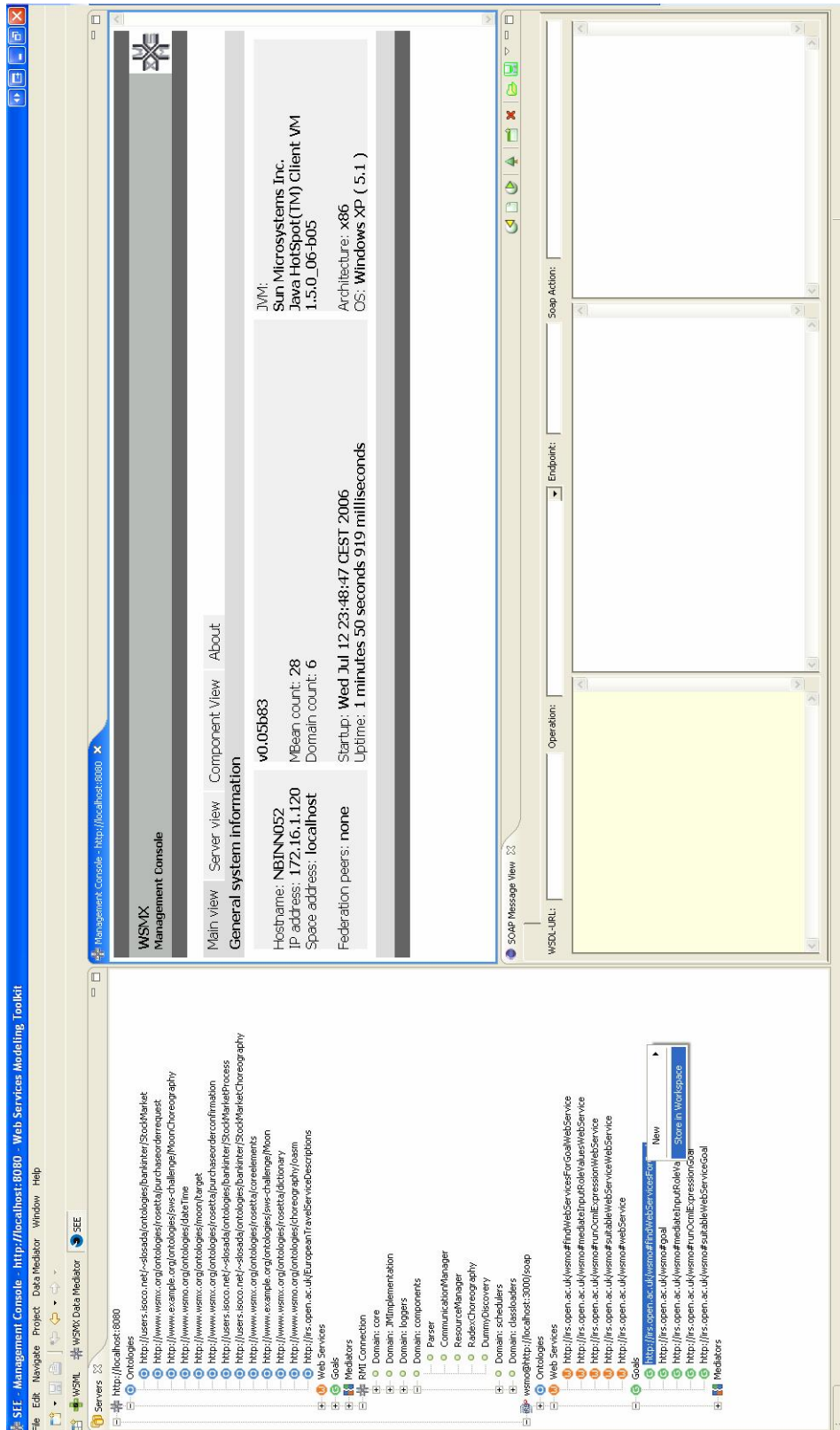


Figure 1: WSMT SEE Perspective

2.6.2 Invoking WSMX as a Web Service

WSMX also exposes its entry points using WSDL via its Axis port (typically 8050 e.g. <http://localhost:8050/axis/services/WSMXEntryPoints?wsdl>)

The WSDL for WSMX can be retrieved from this URL and used to send WSML messages to WSMX either programmatically or through a tool such as StrikeIron⁷.

2.6.3 Examples of WSML Resources

Sample WSML files that can be used with WSMX are available at:

[http://wiki.wsmx.org/index.php?title=Sample WSML](http://wiki.wsmx.org/index.php?title=Sample_WSML)

2.7 Roadmap for future plans

The WSMX prototype will be further developed in the course of the FP6 Integrated Project - SUPER⁸, which is investigating Semantic BPEL through the evolution of the research results of DIP. Further prototype versions will feature new refinements, and add additional components to the existing set developed for WSMX.

3 ARCHITECTURE PROTOTYPE DOCUMENTATION

3.1 Server Administration GUI

The microkernel hosts an HTTP daemon that provides a GUI console for administrative tasks. 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 are 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 2: WSMX Management Console – Main View

As can be seen in Figure 2, the different services, deployed in the WSMX server as MBeans, are grouped by their domains. Besides the domain of the actual WSMX

⁷ <http://www.strikeiron.com/>

⁸ <http://www.ip-super.org/>

components, several other important domains can be seen in the Server view, e.g. Classloaders, Core services and Loggers. Selecting one of these components from the server view opens up the MBean⁹ view of this service, where different service attributes can be manipulated and service methods can be executed.

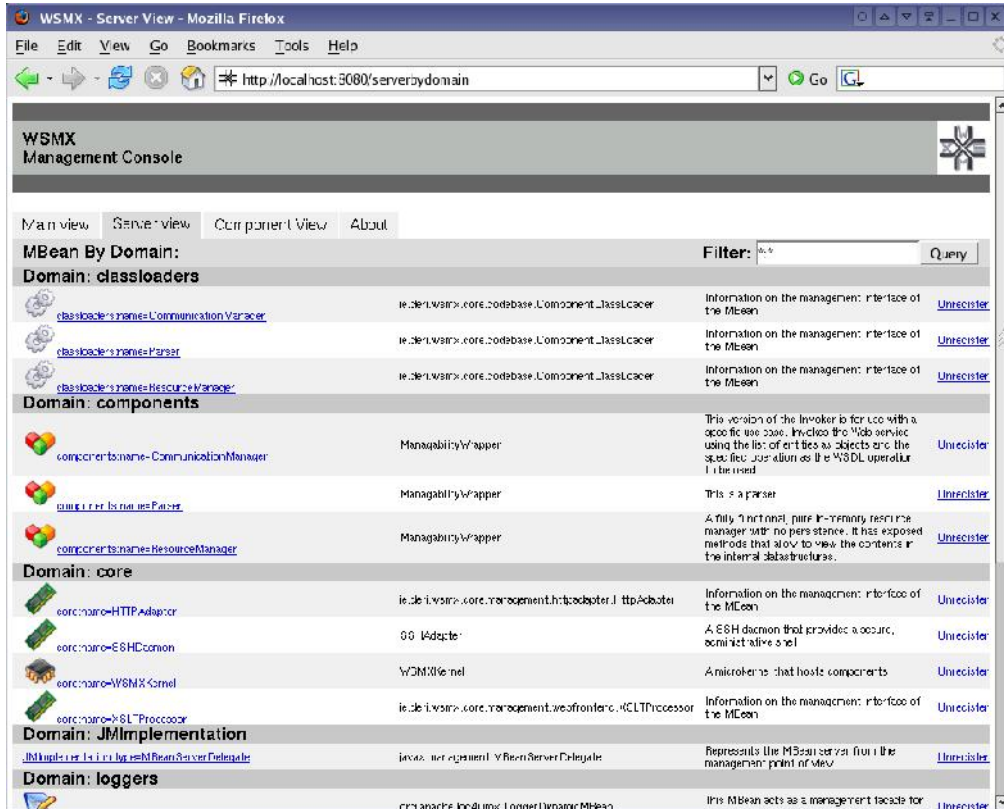
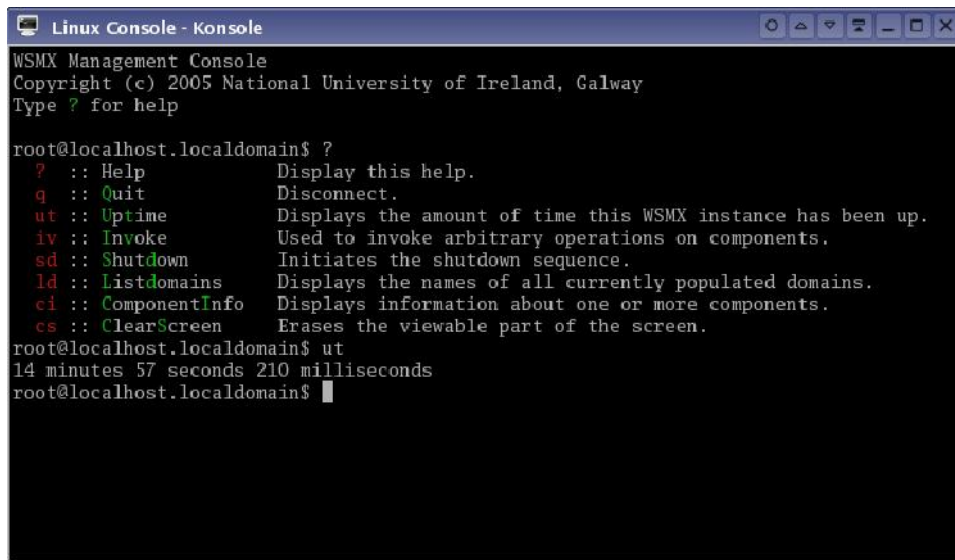


Figure 3: WSMX Management Console – Server View

3.2 Server Administration TUI

Besides the GUI described above, the microkernel hosts an SSH daemon that provides a Terminal User Interface (TUI) console for administrative tasks. Figure 4 shows a screenshot of the TUI, on which the basic commands available can be seen.

⁹ an MBean (managed bean) is a Java object that represents a manageable resource, such as an application, a service, a component, or a device



```
Linux Console - Konsole
WSMX Management Console
Copyright (c) 2005 National University of Ireland, Galway
Type ? for help

root@localhost.localdomain$ ?
? :: Help          Display this help.
q  :: Quit         Disconnect.
ut  :: Uptime      Displays the amount of time this WSMX instance has been up.
iv  :: Invoke      Used to invoke arbitrary operations on components.
sd  :: Shutdown    Initiates the shutdown sequence.
ld  :: Listdomains Displays the names of all currently populated domains.
ci  :: ComponentInfo Displays information about one or more components.
cs  :: ClearScreen Erases the viewable part of the screen.

root@localhost.localdomain$ ut
14 minutes 57 seconds 210 milliseconds
root@localhost.localdomain$
```

Figure 4: 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, Process Mediator, Adapter Framework, Communication Manager, Resource Manager, Choreography Engine, WSMML Flight Reasoner) have had no significant changes since the last version of this deliverable. Their descriptions can be found in [24].

3.3.1 Discovery Engine

Current Version: 0.2

Available at: WSMX CVS at SourceForge (<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 component features quality-of-service (QoS) aspect while performing discovery. The QoS based discovery is described in [25]. The implementation of this feature takes

either keyword or lightweight semantic discovery and filters out those services that do not meet certain QoS characteristics as defined by the Goal.

The component also provides contracting feature as described in [26]. The idea is to allow coexistence of Semantic as well as non-Semantic systems. It uses WSMX as a middleware platform which provides adapters for adapting non-Semantic data e.g. encoded in XML to WSML.

3.3.2 Orchestration Engine

Current Version: 0.3

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

This version of the Orchestration Engine is the continuation of its work as described in [27]. 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 version provided richer supports for *achieveGoal* entry point. This will allow using *achieveGoal* directly from the orchestration. In addition, it provides support for data mediation which might be necessary while orchestrating Web services.

The general information on usage of this component is available at [27].

3.3.3 WSML Flight Reasoner

Current Version: 2.0

Available at: `cvs.deriv.at:/usr/local/cvsroot` (module `wsm2reasoner`)

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 previous described in [27], it handles all constructs of this language variant according to the WSML semantics specification in [19] and in [22].

In this version, new improved implementations are plugged in without altering the original architecture. In this version, the component uses the WSML Reasoner API as defined in [9] and allows pluggability features to plug-in KAON and MINS reasoners. KAON¹⁰ is essentially an OWL-DL reasoner but has been relaxed to work with WSML-DL reasoner. As a fully functional OWL-DL reasoner was already available, DIP did not restrict itself for using it in order to avoid reinventing a wheel. MINS¹¹ is a reasoner for Datalog programs with negation and function symbols. It supports the Well-Founded Semantics and is based on SILRI interface engine.

The deployment package of the WSML-Flight reasoner is also available in [9].

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>

¹⁰<http://kaon.semanticweb.org>

¹¹ <http://tools.deriv.org/mins/>

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 RELATION TO WSMO4J AND WSMO STUDIO

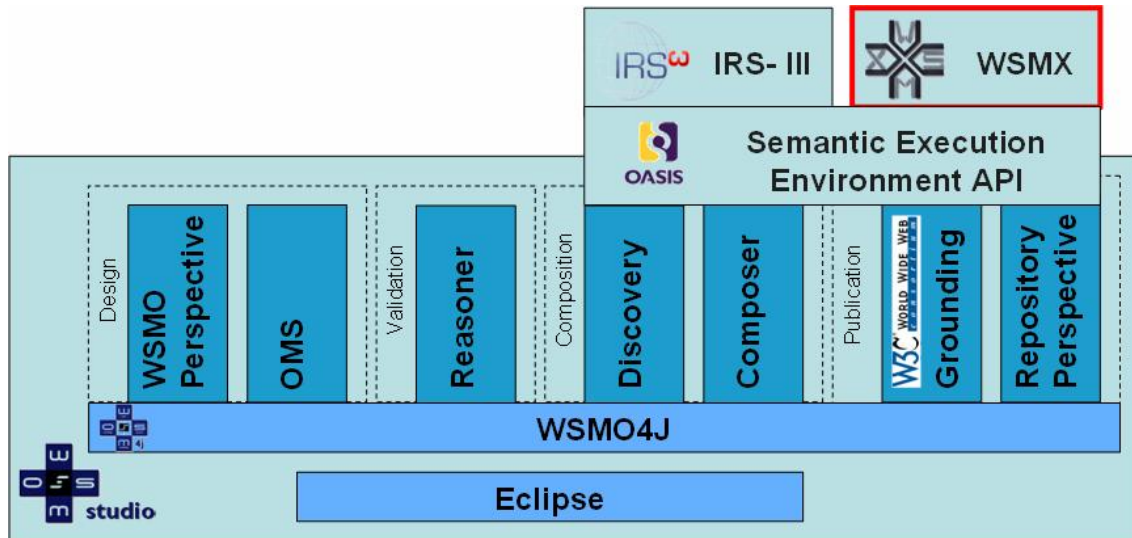


Figure 5: WSMX Relationship to Other DIP Products

Figure 5 illustrate how WSMX is related to the other software artefacts developed in the course of DIP. WSMX provides a prototype implementation of the architecture based on the define DIP (now SEE) API. WSMOJ provides the Java object model used by all DIP tools while WSMO Studio is an eclipse-based environment that hosts the various tools and provides a WSMO editor and management tool.

5 CONCLUSION

This deliverables 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

- [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.; Atanasov, 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.; Balaban, A. (2005): *D6.13: Architecture Prototype v3.0*, 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>

-
- [21] Zaremba, M. et al. (2006): *D6.11: Semantic Web Services Architecture and Information Model*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
 - [22] Jos de Bruijn (editor) (2005): “*D16: The WSML Specification*”, WSML Working Draft 3 February 2005, <http://www.wsmo.org/TR/d16/>
 - [23] 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>
 - [24] Haselwanter, T.; Schreder, B.; Wahler, A.; Zaremba, M. (2005): *D6.13: Software Deliverable and Installation Guidelines v3.0*, DIP Project, WP6 Interoperability and Architecture. <http://dip.semanticweb.org>
 - [25] Hauswirth, M.; Porto, F.; Vu, Le-Hung. (2006): *D4.17: P2P and QoS-enabled service discovery specification*, DIP Project, WP4 Service Usage. <http://dip.semanticweb.org>
 - [26] Zaremba, M.; Vitvar, T.; Moran, M.; Hasselwanter, T. (2006): *WSMX Discovery for SWS Challenge*, SWS Challenge 2006, Stanford University
 - [27] Hasselwanter, T.; Bhiri, S.; Goyal, A. (2006): *D4.20: DIP Orchestration Prototype*, DIP Project, WP4 Service Usage. <http://dip.semanticweb.org>