



D22v0.1 WSMX Documentation

WSMO Working Draft 10 December 2004

Final version:

<http://www.wsmo.org/2004/d22/v0.1/20040826/>

Latest version:

<http://www.wsmo.org/2004/d22/v0.1/20041210/>

Previous version:

<http://www.wsmo.org/2004/d22/v0.1/20040826/>

Editor:

David Aiken

Authors:

David Aiken, Maciej Zaremba

This document is also available in non-normative PDF version.

Copyright © 2004 DERI®, All Rights Reserved. DERI liability, trademark, document use, and software licensing rules apply.

Table of contents

1. Introduction
 - 1.1 Purpose of this Document
 - 1.2 Document Overview
2. WSMX Installation
 - 2.1 CVS
 - 2.2 WSMX Dependencies
 - 2.3 J2SDK

- 2.4 JWSDP
 - 2.5 Eclipse-SDK
 - 2.6 MySQL
 - 2.7 Apache-Ant
 - 3. WSMX Usage
 - 3.1 WSMX as Event Driven Architecture
 - 3.2 WSMX Entry Point
 - 3.3 WSMX Components
 - 3.3.1 Adapter Framework
 - 3.3.2 Choreography
 - 3.3.3 Mediator
 - 3.3.4 Parser/Compiler
 - 3.3.5 Discovery
 - 3.3.6 Communication Manager
 - 3.3.7 Resource Manager
 - 3.3.8 Core Components
 - 4. Conclusions and Future Work References Acknowledgment
-

1. Introduction

The Web Services Modeling Execution Environment (WSMX) is an execution environment for dynamic discovery, mediation and invocation of web services. WSMX is based on the Web Services Modeling Ontology (WSMO) [Roman et al., 2004], an ontology for describing various aspects related to Semantic Web Services. So far Web Services are mostly hard-wired into the provider's software and their abilities are not too sophisticated. Their main drawback is lack of semantic description and because of that it is difficult to perform automatic or semi-automatic service retrieval. WSMO aims to change this situation in order to fully enable Web Service functionality through adding semantic means that will result in Web Services vast proliferation. WSMX is a reference implementation and test bed environment for WSMO. Web Services Capability Description are stored in the WSMX repository and they are described in terms of logic expressions so they can be derived from. When service request (Goal) is sent to WSMX, then WSMX components start to perform their actions. First, the message is parsed, then it is matched against WS Capability, then if required mediation is performed in order to mitigate differences between different ontologies. Afterwards WSMX selects the most relevant WS and finally invokes it. One of the implementation goals is to keep components loosely coupled in order to easily add, remove or change them.

1.1 Purpose of this Document

This document provides the description of a step by step WSMX installation and examples of its usage. It starts with an installation guide and in the next section explains implementation details, especially the event architecture approach.

1.2 Document Overview

Section 2 gives a guide of WSMX installation. Section 3 provides implementation details. Section 4 concludes the document and looks at future work.

2. WSMX Installation

In this section we explain a step by step WSMX installation. WSMX is currently available on the CVS on the SourceForge server (WSMX hosting site - <http://sourceforge.net/projects/wsmx/>).

Following software is necessary to run WSMX:

- J2SDK.
- J2EE or Tomcat.
- MySQL Server (database system), MySQL Control Centre and MySQL ConnectorJ.
- Eclipse and Ant.

2.1 CVS

You must have a developer status in order to have a right to commit code on CVS repository. To get a developer status contact the project supervisor. Once you have been granted developer status you need to generate access keys. Details on how to do it can be found on the SourceForge website. There is also anonymous read-only access provided.

To get sources from SourceForge CVS repository create a new CVS repository in Eclipse. Add a SourceForge WSMX CVS repository. Next, you can create either an empty Eclipse project or directly create a CVS project and specify the CVS location. Default location specified in WSMX properties files is **d:\development\webapp\wsmx**. If you are going to use a different location then this path should be changed in WSMX dependencies (described in point 2.3).

2.2 WSMX dependencies

Following System Environment Variables should be set:

- JAVA_HOME - e.g. **c:\Java\j2sdk**
- CLASSPATH - e.g. **c:\Java\j2sdk\lib**
- J2EE_HOME - e.g. **c:\Java\j2ee**
- WSMX_HOME - e.g. **d:\development\webappl\wsmx**

WSMX configuration files:

- \$(USER_HOME)\wsmx.properties - specifies connection to JDBC, login/password, paths to log4j properties files, etc.,
- \$(WSMX_HOME)\build.properties - specifies paths to JDBC driver libraries,
- \$(WSMX_HOME)\bin\config.bat - specifies paths to J2SDK, J2EE and WSMX directories,
- \$(WSMX_HOME)\bin\startWSMX.bat - starts WSMX,

2.3 J2SDK

Step 1.

Go to the following address:

<http://java.sun.com/j2se/1.5.0/download.jsp>

for downloading Java(TM) 2 SDK, Standard Edition 1.5.0.

Step 2.

Run the installer *j2sdk-1_5_0-windows-i586-p.exe* you have just downloaded.

License Agreement:

Follow the link from the Step 1 to find the license agreement.

2.4 JWSDP

Installation

Step 1.

Download *Java WEB Services Developer Pack 1.5* from:

<http://java.sun.com/webservices/downloads/webservicespack.html>

Step 2.

Run the Java WSDP installer (*jwsdp-1_5-windows-i586.exe*).

Step 3. (optional)

You can run the Java WSDP without setting the PATH variable, or you can optionally set it as a convenience.

License Agreement:

Follow the link from the Step 1 to find the license agreement.

The license agreement can also be found in `..\jwsdp-1.3\LICENSE`

Configuration

Choose Start, Settings, Control Panel, and double-click System. Select the Advanced tab and then Environment Variables. Look for "Path" in the User Variables and System Variables. If you're not sure where to add the path, add it to the right end of the "Path" in the User Variables. A typical value for PATH is:

`C:\jwsdp-1.5\bin`

Capitalization doesn't matter. Click "Set", "OK" or "Apply".

The PATH can be a series of directories separated by semi-colons (;). Microsoft Windows looks for programs in the PATH directories in order, from left to right. You should only have one bin directory for a Java WSDP installation in the path at a time (those following the first are ignored), so if one is already present, you can update it.

Some WSMX components are running as Web Services (e.g. compiler, adapters) that are deployed using Apache Axis libraries that are compiled into WSMX. In order to grant them appropriate rights to run on J2EE server, copy the content of **\$(WSMX_HOME)\doc\server.policy** to the end of **\$(J2EE_HOME)\domains\domain1\config\server.policy**. Particular policies for deployed application on J2EE server must be specified so you can restrict or grant permissions to system resources and system functions (e.g. write file, connect, resolve for particular java libraries etc.).

2.5 Eclipse-SDK

Step 1.

Download *eclipse-SDK-3.0.1* from:

<http://www.eclipse.org/downloads/index.php>

Step 2.

Unpack the *eclipse-SDK-3.0.1-win32.zip* file and run *eclipse* application file.

License Agreement:

The license agreement can be found in `..\eclipse\notice.html`

2.6 MySQL

Installation

MySQL Server 4.1

<http://dev.mysql.com/downloads/mysql/4.1.html>

MySQL Control Center

<http://dev.mysql.com/downloads/other/mysqlcc.html>

MySQL ConnectorJ 3.1

<http://dev.mysql.com/downloads/connector/j/3.1.html>

License Agreement:

The license agreement can be found in at:

<http://www.mysql.com/products/licensing.html>

Configuration

- create new database: wsmx.
- add new user login/pass: wsmx/wsmx, host: localhost. User should have all privileges to wsmx database.

2.7 Apache-Ant-1.6.2

Step 1.

Download Apache-Ant-1.6.2 from;

<http://ant.apache.org/srcdownload.cgi>

Step 2.

Follow instructions <http://ant.apache.org/manual/index.html>

License Agreement:

The license agreement can be found in at:

<http://ant.apache.org/manual/index.html>

Open Ant view in Eclipse. Load build.xml file from \$(WSMX_HOME). There are many dependencies between ant entities (i.e. one entity invokes other one). Here is a list of most relevant ant entities:

- clean - removes build directory, it can be used to make sure that files included in compiled version are up-to-date, however, the java compiler checks during every compilation if any *.java files were changed,
- createdb - runs SQL script to create required database tables,
- cleandb - removes all data and tables from database,
- build-wsmx-manager - compiles wsmx-manager component,
- deploy-web-services - compiles and deploys current version of WSMX,
- runTestApplication - runs a back-end application,

Start JWSD server. If all paths and dependencies are set properly run deploy-web-services Ant entity. To start all WSMX components run \$(WSMX_HOME)\bin\startWSMX.bat in the command prompt. If no errors were reported, WSMX listeners are ready to pick up incoming events. To stop WSMX

components properly run `$(WSMX_HOME)\bin\stopWSMX.bat`. WSMX creates a `.pid` file in `$(USER_HOME)\WSMX\`. Database guarantees persistency and WSMX creates an additional file with its current state. In case of a system crash this file is intended to be used for recovery, however, in the current version it is a "dummy" file. WSMX Manager just checks whether this file exists. If WSMX was closed properly the `*.pid` file is deleted, otherwise WSMX Manager refuses to start and user should delete this file manually or run `stopWSMX.bat`.

3. WSMX Usage

3.1 WSMX as Event Driven Architecture

In this section we provide WSMX Usage description and an overview of WSMX approach to handling incoming messages and its internal workflow policy.

WSMX is a Event Driven Architecture. This means that there are no direct invocation between components. Each component is subscribed to the WSMX Manager listener and has its **type** and **status**. Type specifies data type of the Event (e.g. `WSML_MESSAGE`, `NON_MEDIATED_OBJECT`, `MEDIATED_OBJECT`). Examples of event's status are: `BEFORE_PARSING`, `AFTER_PARSING`, `ERROR`, `CONSUMED` etc. Each component expects certain Event status and type respectively. When new Event appears in system then all components that are subscribed as a listeners receive information that there is an Event to pick up. There is a notion of double lock. When component enters its critical path (i.e. check what is the type and status of an awaiting Event) then the first lock is imposed in order to avoid checking this Event by other components. If appropriate then the second lock is imposed and it means that the particular Event is being processed by a component.

3.2 WSMX Entry points

There are two entry points to WSMX:

- WSMX Manager - receives WSML goal from the back-end application, then it creates an Event for this message that is picked up next by the WSMX components. Entry point is depicted on the Figure 1 this on left hand side. If the back-end application output format is other than WSML, it is necessary to send it first to the adapter that performs a format conversion and then forwards a WSML message to WSMX manager.
- WSMX Compiler - receives Web Service Capability Description from WSMX Editor and if appropriate adds this Capability to its repository. WSMX-Compiler is a Web Service application, is does not create an Event. It is depicted

as topmost element on the following picture,

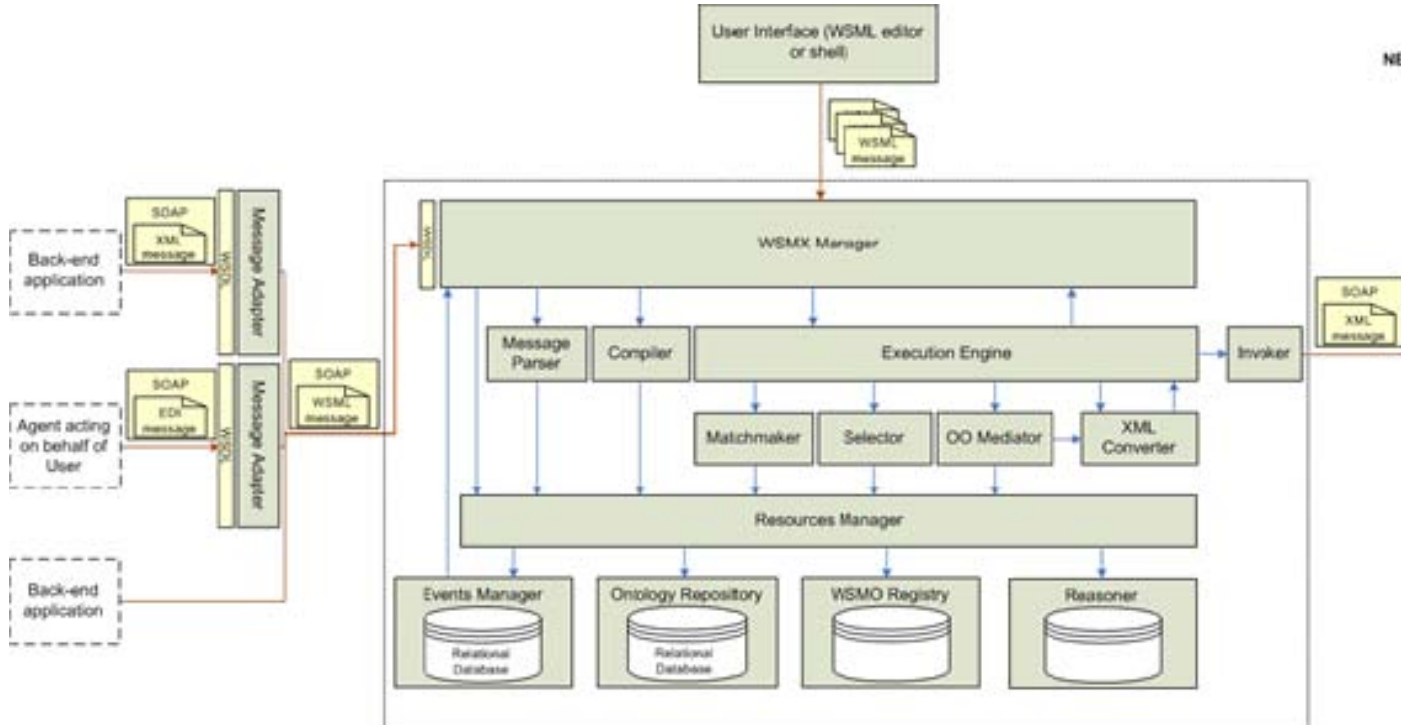


Figure 1: WSMX components architecture overview

3.3 WSMX Components

This section details each of the components listed in the architecture and describes them under three headings; 1) Introduction, 2) Functionality and 3) how to run a test program.

3.3.1 Adapter Framework

1. Introduction

Adapters reside as an intermediary between incompatible systems and facilitate communication between them. To enable compatibility an adapter converts data presented in one format to another. Depending on the system and business application, adapters can be used individually or grouped in clusters for more complex usage. The clusters are designed as frameworks where adapters can be plugged -in or -out according to the communication needs.

2. Functionality

On the conceptual level, an adaptor transforms the received message from its source format to the target format. The transformation is concerned only with the syntactic mapping between message formats, leaving the semantics of the message intact. The semantic part is enforced by personalized ontologies that are used to build the outgoing message format.

The main problem tackled by the WSMX adaptor framework is the transformation from UBL (XML) into the language comprehensible by WSMX (WSML). The syntactic mapping between ontologies in different formats is done by extracting the instances from the source ontology fragment, then mapping them and finally populating the target WSML ontology fragment.

3. Test program

A new component emulates WSMX behavior (fake WSMX) in receiving and replying back messages to the adaptor framework. Test messages are sent in XML via the back-end application; XML2WSML adapter transforms this XML into WSML and forwards the message to the fake WSMX. The fake WSMX replies back a WSML message following the logic hard coded in it for that specific test. WSML2XML Adapter transforms back the received WSML to XML and forwards it to the back-end application.

3.3.2 Choreography

1. Introduction

A choreography defines how to interact with the web service in terms of exchanging messages, the so called communication patterns. It describes how a requester can interact with the service provider (in the case of a WSMX execution a Web Service). The requestor of the service has its own communication pattern and only if the two of them match precisely, a direct communication between the requestor and the provider of a service may take place.

By matching precisely we understand the full matching of the communication pattern from the source and target parties, also called process equivalence. That means that for each possible instance of the source choreography at least one instance of the target choreography is available.

Since usually the client has its own communication pattern that in general is different from the one used by the Web Service, the two of them will not be able to directly communicate, even if they are able to understand the same data formats. In order to invoke the Web Service the two parties must either redefine their communication patterns (or at least one of them) or to use an external mediation system as part of the process. The second approach is exactly what the WSMX choreography engine tries to achieve.

The role of the Choreography Engine is to put together the necessary means for the runtime analyses of two given choreography instances and to use the

mediators to compensate the possible mismatches that may appear, for instance, to generate dummy acknowledgement messages, to group several messages in a single one, to change their order or even to remove some of the messages in order to facilitate the communication between the two parties. The above presented functionality is based on a design time process which identifies the equivalences between the choreographies' conceptual descriptions, that is, a set of rules are created and stored, in order to be later applied on the particular choreography instances, during runtime.

2. Functionality

In the first release of the Choreography component we intend to provide a simplified matching of correspondent nodes of stored Choreographies. Nodes are defined by a databinding, a nextnode parameter and an identifier. For the first version there are no dependencies on other components. This will change when mediation will be considered to overcome process heterogeneities. Currently no mediation is performed, which implies that no communication can take place if the choreographies of the provider and requester are not equal.

3. Test program

There is no Test program implemented yet.

3.3.3 Mediator

1. Introduction

WSMX Data Mediation Component has the role of reconciliation the data heterogeneity problems that can appear during discovery, composition, selection or invocation of Web Services. This component offers support for both the runtime phase and design time phase of the mediation process. That is, the design-time subcomponent offers a graphical interface that assists the user in the creation of mappings and the run-time component makes use of the already created mappings and performs the actual data transformation in a completely automatic manner. In fact, only the second subcomponent is explicitly part of the WSMX architecture, but its functionality is enabled by the outputs of the design-time component.

2. Functionality

Design-time component

This component consists of a graphical interface that offers support to the domain expert placing its inputs (i.e. mappings). The user can browse the ontology and can decide, based on the offer suggestion, what 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 main dependencies of this component are on the lexical analyser level and on the storage level. For the lexical analyser a critical requirement is that WordNet has to be installed on the machine where the mediation component runs. As for the storage used by the mediation component for saving the mappings, it is in the current version an independent relational database (MySQL Server is used). For the future versions some of the functionality offered by this storage will be covered by WSMX persistency layer, and as a consequence, the mediation storage will be the subject of important restructurings.

Run-time Component

The run-time component has the role to retrieve from the storage the already created mappings, to transform them in rules and finally to execute them against the incoming instances in order to obtain the target instances.

Since the mappings represents the connection point between the two sub-components (design-time and run-time) one of the dependencies for run-time component is on the mapping storage level as well.

Another crucial dependency is the reasoning system used for executing the rules in the final stage of the mediation process. For this, Flora2 engine together with its underlying system, XSB Prolog, are used; they are integrated in Java by using InterProlog, a Java front-end and enhancement for XSB Prolog. The installation of this system is completely handled by a set of scripts coming with the mediation component, without any burdensome from the user side.

3. Test program

The design time component is meant to be run as a stand-alone applications (as is mentioned in a previous section, it is not explicitly part of the WSMX architecture). The component can be started by running:

```
ie.deri.wsmx.mediation.ooMediator.newgui.RunMe.java
```

On the other hand, the runtime component is explicitly part of the WSMX architecture and it is invoked by WSMX each time data mediation is required. In addition the design-time component contains a special panel that allows you to simulate a WSMX invocation on the runtime component. In this way you can very easy test this component and also the results of the mappings that have been just created.

3.3.4 Parser/Compiler

1. Introduction

The parser performs syntactic validity checks of WSML documents provided by the execution manager. It determines whether a WSML document can be processed and converts it into an internal representation.

The parser makes use of the WSMO API, which defines methods for validating a document against a certain WSML variant and returning a memory-model of the elements of the document. The WSMO API is complemented by a reference implementation, which includes a parser to check the validity of documents against a certain WSML variant.

The reference implementation returns an in-memory model of the elements of the parsed document. This in-memory representation is passed to the DBManager for persistent storage.

2. Functionality

- parses wsml file and stores content in persistent storage

depends on:

- wsmo4j parser for parsing
- dbManager for storage.

3. Test program

run main method in AlphaParser.java.

3.3.5 Discovery

1. Introduction

Discovery Component of WSMX will provide a full implementation of discovery mechanism provided in WSMO Discovery **Error! Reference source not found..** This component is developed within the deliverable D5.2: WSMO Discovery Engine **Error! Reference source not found..** Different tools and formalisms are tested to see which of them are more suitable for what is needed. For keyword-based discovery described in D5.2 a simple implementation is already available and integrated with WSMX. Future plans for implementation include the distributed discovery.

2. Functionality

In this section the implementation of keyword-based discovery functionality is provided. Two methods can be used if the keyword-based discovery functionality is required. These are: *matchByGlobaNFP* and *matchByAxiomNFP*.

The *matchByGlobaNFP* method performs a keyword-based match over the NFP of the goal description and web services descriptions. Given a Goal, a set of known Web Services and a KeywordDiscoveryPreference this method returns a set of match services. A **KeywordDiscoveryPreference** contains the *NFP attribute* over which the keyword search is performed, the *type of match* (*Full match* or *Partial match*) and the *threshold*, which is used only if the type of match is *Partial*. For the Full match the whole value of the NFP attribute from the goal is compared with the whole value of the each web service. For the Partial match

individual keywords from the goal NFP are compared with initial keywords from the web services NFP and a score is computed for each service. These score is normalized and then in the selection process only the normalized scores that have a bigger value then the given threshold are selected.

The ***matchByAxiomNFP*** method performs a keyword-based match over the NFP of the axioms that describe goal postconditions and NFP of the axioms that describe service capabilities and postconditions. The algorithm is the same as in ***matchByGloboNFP*** method but applied to NFP attributes of the axioms that define the postconditions of the goal and web services.

The next step is to investigate how distributed discovery can be performed in WSMX. In order to discover other WSMXes we envision three fundamental approaches. The first one is an UDDI 0 like approach that uses a central repository where different WSMXes are registered. The second approach uses a P2P infrastructure. Finally the third approach is based on Triple Space paradigm **Error! Reference source not found.** We think that the last two approaches are more suitable for WSMX distributed discovery, the UDDI approach being a too centralize approach.

For the next version of the Discovery component we will investigate and implement P2P distributed discovery for WSMX. We see two options to build the P2P infrastructure: the first one is based on JXTA **Error! Reference source not found.**, the second one, which we call it “Hypercube” approach is described in **Error! Reference source not found.** On top of the P2P network we will implement the distributed discovery mechanism.

2. Test program

To test the keyword-based discovery a simple JTest Unit is provided. This program test can be as a standalone program, and it dose not required any parameters. Using the `wsmo4j` we create the goal and web services java objects that are needed for the discovery process. The functionality of ***matchByGloboNFP*** and ***macthByAxiomNFP*** is tested. These methods return a set of matched services given a goal and a set of known services. The test program returns how many web services were matched for each method.

3.3.6 Communication Manager

1. Introduction

The Communication Manager is responsible for sending and receiving WSML messages to/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 can be invoked by WSMX adapters are expected to be used where the external entity does can not 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'. The two central functionalities of the Communication Manager - sending and receiving messages - are described in the next sections.

2. Functionality

Sending Messages

The Communication Manager provides an interface to receive the WSMO description of the service to be invoked along with data to be sent in the message to the service. The data must consist of instances of concepts described in the ontology used by the service. Both the service description and data will be represented in WSML. The Communication Manager interprets the interface part of the service description to determine which binding is necessary to invoke the service or, the adapter representing the service, where one is required.

Receiving Messages

The Communication Manager provides an interface to external entities to accept WSML messages. The WSML messages may represent a goal to be achieved by another WSMX instance or be a message corresponding to a choreography or orchestration instance that already exists. From the perspective of the Communication Manager, the distinction is irrelevant. The Communication Manager accepts the message, handling any transport and security protocols used by the sending adapter.

3. Test program

The Communication Manager will be provided with a set of unit tests that can be run using the JUnit (<http://www.junit.org>) test framework.

3.3.7 Resource Manager

1. Introduction

The WSMX Resource Manager is responsible for storing every data WSMX uses. Because WSMX is developed with the programming language Java and every data is represented as a Java object, the resource manager uses an O/R-Mapper for storing the data. An O/R-Mapper provides functionality to map an object oriented data model to a relational data model, i.e. a database.

2. Functionality

The domain model of WSMX is based on the WSMO API which is just a set of interfaces representing the data structure of WSMO 1.0. The Resource Manager provides an implementation of those interfaces which should be used to transfer data between every WSMX component. The implementation is independent of the O/R-Mapping.

The O/R-Mapper which is used by the resource manager is Hibernate

(<http://www.hibernate.org>). The information how Hibernate should map the Java objects to the database is stored in XML files. Out of those files Hibernate is managing the whole database, the structure of the tables and the data itself. Besides Hibernate the resource manager uses the Springframework (<http://www.springframework.org>) to manage the connection to Hibernate. Spring is a lightweight container which provides a generic access to different O/R-Mapping tools and strategies. This makes it easier to switch to other tools or strategies without influencing the application.

For accessing the data/objects with data from the database the resource manager uses the Data Access Object (DAO)– design pattern. This means that there is for almost every object of the domain model a class with store/update and several find-methods. On top of those DAOs is a single manager facade which simply provides the access to every DAO. This is the single entry point/interface to the resource manager. So for loading/storing the objects of the domain model only the interface of the manager facade should be used.

3. Test program

There is a simple client class with a main method which generates some objects, tries to store and update them and then load them again from the database. Besides this every DAO has its own Junit test case which tests every functionality provided by this DAO.

3.3.8 Core components

Pending...

4. Conclusions and Future Work

This document presents the general overview of the WSMX first release Implementation Installation and Usage. Although first WSMX version remains incomplete in terms of its functionality, it is already able to pass incoming WSML messages throughout the system and match simple goals. A great effort has to be put into further development of the WSMX components. There are ongoing works on developing expressivity of WSML language and its reasoner. Most crucial parts in current version are Matchmaker (demands logic reasoning) and OOMediator (import ontologies and resolve possible representation mismatches between ontologies). In current implementation there is only OOMediator for simplicity reason, whilst in WSMO [Roman et al., 2004] there are many more Mediators described. In next WSMX version new components will be introduced. There might be the following components:

- **GGMediator** - mediators that link two goals. This link represents the refinement of the source goal into the target goal,
- **WGMediator** - link Web Service to goal,
- **WWMediator** - link two Web Services,
- **MessageManager** - in current version communication is only one-way, in order to establish two-way communication we need a special component. There are additional issues with regard to communication, for example, a return message may need to be mediated and/or send through some dedicated adapter if back-end application doesn't speak WSML language,
- **CorrelationManager** - in some cases in order to execute Web Service some conditions must be fulfilled, for example, it is required to check if telecom provider can provide ADSL line for a particular phone number before ordering a line (see Niva use case). So new sub-events need to be created in order to determinate whether execution conditions are fulfilled or not. The main Event should be put to sleep and when the sub-events are finished, it should be woken up and check if the sub-events results allows him to invoke Web Service,

Additionally WSMX should use WSIF (Web Services Invocation Framework) in order to separate invocation from technical issues (e.g. SOAP message document style or RPC-style).

In the current version, execution semantics is hard-wired in Java code. Each component expects particular Event type and status, and execution logic (i.e. condition branching) is coded in Java as well. In order to make execution semantics more flexible we introduce a workflow engine to run WSMX based on sound theoretical foundations (e.g. Petri Nets). It could be YAWL [van der Aalst et al., 2004] , it has a good theoretical basis (i.e. one can simulate and analyze any potential deadlocks). This approach would make WSMX semantic execution far more flexible, however, it also needs to create a proper interface for objects. One of the most important demands for workflow engine is asynchronous execution. If possible, workflow has to switch between transitions (invoking components) to avoid awaiting for particular components to responded.

References

[Bellwood et al] Bellwood, T., Climent, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y., Januszewski, K., Lee, S., McKee, B., Munter, J., and von Riegen, C. (2002). UDDI version 3.0. <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>.

[Bussler, 2003] C. Bussler, B2B Integration, Concepts and Architecture, Springer-Verlag, 2003, ISBN 3-540-43487-9

[van der Aalst et al., 2004] W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede, Design and implementation of the YAWL system, [http://www.citi.qut.edu.au/yawl/ docs_for_yawl/yawls.pdf](http://www.citi.qut.edu.au/yawl/docs_for_yawl/yawls.pdf)

[ebMS, 2002] Message Service Specification, Version 2.0, OASIS ebXML Messaging Services Technical Committee, 1 April 2002, <http://www.ebxml.org>

[Fensel, 2004] Fensel, D. (2004). Triple-based Computing. Technical report, Digital Enterprise Research Institute (DERI). Look at <http://www.deri.org/TR/2004-05-31>.

[Herzog et al., 2004] R. Herzog, H. Lausen, D. Roman, M. Stollberg, P. Zugmann. WSMO Registry, <http://www.wsmo.org/2004/d10/v0.1/>

[Keller et al., 2004a] Keller, U., Lara, R., Polleres, A., Toma, I., Killer, M., and Fensel, D. (2004). Web Service Modeling Ontology – Standard (WSMO-Standard). Working draft, Digital Enterprise Research Institute (DERI). Available from <http://www.wsmo.org/2004/d5.1/v0.1>.

[Keller et al., 2004b] Keller, U., Lara, R., Polleres, A., Luassen, H., Predoiu, L. and Toma, I. (2004). WSMO Discovery Engine. Working draft, Digital Enterprise Research Institute (DERI). Available from <http://www.wsmo.org/2004/d5.2/v0.1>.

[Oren, 2004] E. Oren. WSMX Execution Semantics, [http://www.wsmo.org/2004/d13/d13.2/ v0.1/20040531/index.pdf](http://www.wsmo.org/2004/d13/d13.2/v0.1/20040531/index.pdf)

[Oren et al., 2004] E. Oren, M. Kifer, J. de Bruijn, H. Lausen, D. Roman, M. Felderer, BNF grammar for WSML user language, <http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418>

[Roman et al., 2004] D. Roman, H. Lausen, U. Keller. Web Services Modeling Ontology -Standard (WSMO - Standard), <http://www.wsmo.org/2004/d2/v02/20040306/>

[RosettaNet] The RosettaNet consortium for open eBusiness standards and

services, [http:// www.RosettaNet.org](http://www.RosettaNet.org)

[Schlosser et al] Schlosser, M., Sintek, M., Decker, S., and Nejd, W. A Scalable and ontology-Based P2P Infrastructure for Semantic Web Services.

[Stollberg et al., 2004] M. Stollberg, H. Lausen, A. Polleres, R. Lara, U. Keller, M. Zaremba, D. Fensel, M. Kifer. WSMO Use Case Modeling and Testing, [http://www.wsmo.org/2004/d3/d3.2/ v0.1/20040524/](http://www.wsmo.org/2004/d3/d3.2/v0.1/20040524/)

[Vasiliu et al., 2004] L. Vasiliu, M. Moran, C. Bussler, WSMO in DIP, WSMO Working Draft v0.1, 7 June 2004, Digital Enterprise Research Institute, available from <http://www.wsmo.org/2004/d19/d19.1/v0.1/>

[W3C Note, 2001] Web Services Description language (WSDL) 1.1, W3C Note 15 March 2001, <http://www.w3c.org/TR/wsdl>

Acknowledgment

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, and SWWS; by Science Foundation Ireland under the DERI-Lion project; and by the Austrian government under the CoOperate programme.

The authors would like to thank to all the members of the WSMO working group for their advises and inputs to this document.



[webmaster](#)