



# D13.4v0.1 WSMX Architecture

WSMO Working Draft 30 May 2004

**This version:**

<http://www.wsmo.org/2004/d13/d13.4/v0.1/20040530/>

**Latest version:**

<http://www.wsmo.org/2004/d13/d13.4/v0.1/>

**Previous version:**

<http://www.wsmo.org/2004/d13/d13.4/v0.1/20040426/>

**Editor:**

Michal Zaremba

**Authors:**

Matthew Moran

Michal 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](#)
  - 1.2 [Purpose of this Document](#)
  - 1.3 [Scope of WSMX Architecture](#)
  - 1.4 [Document Overview](#)
2. [High Level Overview](#)
  - 2.1 [Compilation](#)
  - 2.2 [Execution Functionality](#)
3. [Black Box Overview](#)
4. [Architecture Components and Interfaces](#)
  - 4.1 [Architecture](#)
  - 4.2 [Components of WSMX Architecture](#)

[4.2.1 User Interface - WSML Editor](#)

[4.2.2 WSMX Manager](#)

[4.2.3 Compiler](#)

[4.2.4 Message Parser](#)

[4.2.5 Resource Manager](#)

[4.2.6 Ontology Repository](#)

[4.2.7 Events Manager](#)

[4.2.8 WSMO Registry](#)

[4.2.9 Execution Engine](#)

[4.2.10 Matchmaker](#)

[4.2.11 Selector](#)

[4.2.12 Mediator & XML Converter](#)

[4.2.13 Invoker](#)

[5 . Conclusions and Future Work](#)

[References](#)

[Acknowledgment](#)

# 1. Introduction

The Web Services Modelling Execution Environment (WSMX) is an execution environment for dynamic discovery, selection, mediation and invocation of web services. WSMX is based on the Web Services Modelling Ontology (WSMO) [Roman et al., 2004] which describes all aspects related to this discovery, mediation, selection and invocation. The execution operations of WSMX are made possible by describing elements relating to web services using the Web Service Modelling Language (WSML) [Oren et al., 2004] and then compiling these descriptions into WSMX.

## 1.1 Purpose of this Document

This document provides the software architecture for WSMX. It starts with a high level picture of the WSMX components and how they relate to each other and goes on to provide detailed descriptions of the components, the interfaces each component provides and the data representation of information passing into and out of these components. One of the key design aims for WSMX is keep individual components decoupled from each other. Interfaces should remain stable while it should be possible to replace component implementation as long as the new implementation supports the existing interface.

## 1.2 Scope of WSMX Architecture

There are two operational aspects considered by the WSMX Architecture – compilation and execution. This document describes both aspects in terms of component behaviour. Building on

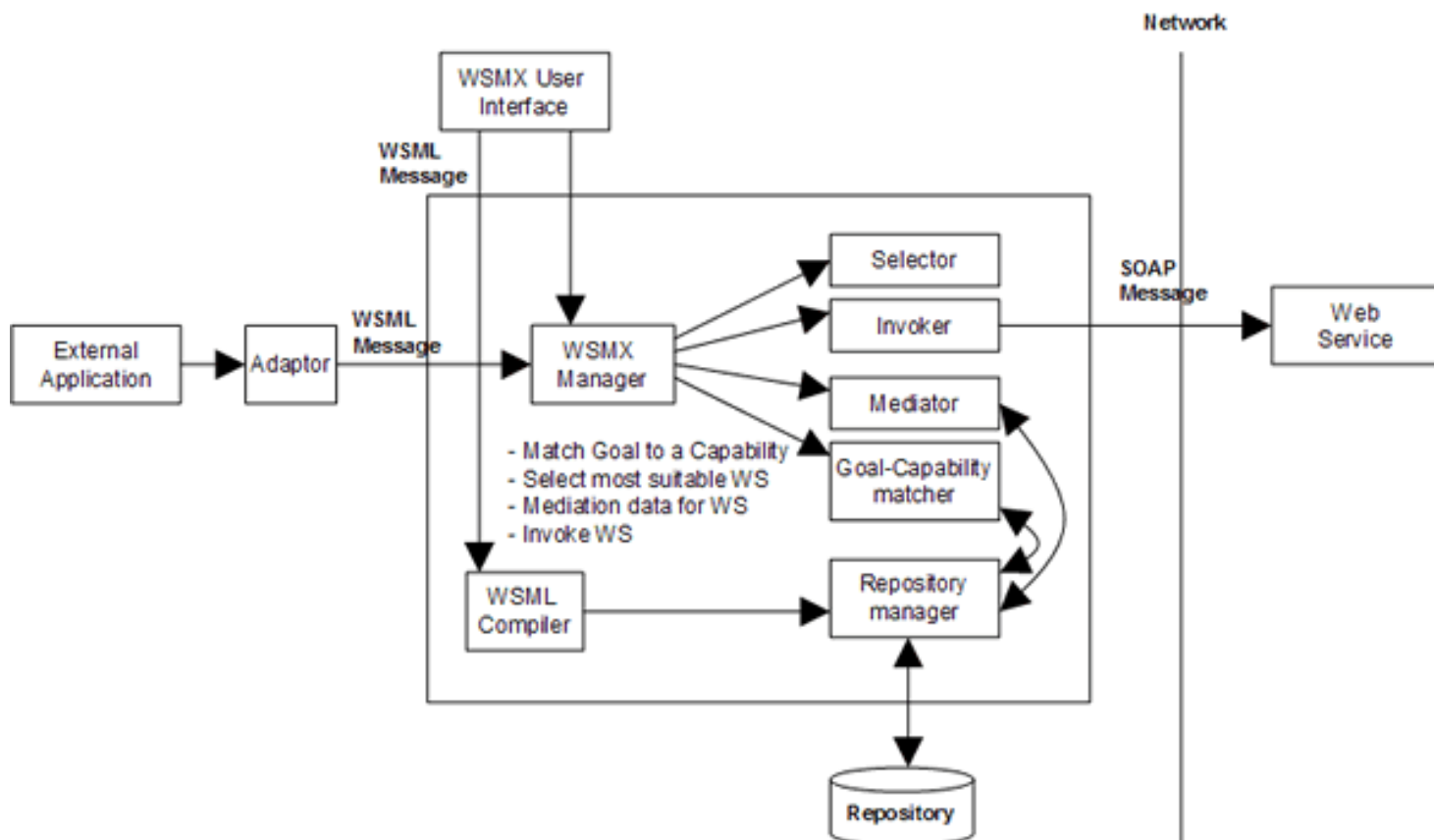
the other deliverables of the WSMX working group, this document provides the specification for the implementation of the WSMX software.

## 1.3 Document Overview

Section 2 gives a high level view of the architecture illustrated with a simple example. Section 3 provides a black box look at WSMX. Section 4 provides details of the architectural components and interfaces starting with an overview of the system and then giving detailed information on each component including event and data representation, state management and definition of the execution flow for the architecture. Section 5 concludes the document and looks at future work.

## 2. High Level Overview

Figure 1 provides an initial overview of WSMX. Aspects of web services are described in terms of WSMO using the User Interface component and passed to WSMX for compilation and storage. The figure also illustrates how an existing backend application would provide its input to an adaptor which would formulate this data into a WSML message containing a description of the goal the application wishes to achieve. The WSMX manager processes the goal based on the execution semantics of WSMX defined in [Oren, 2004]. This involves matching the goal to a capability, finding web services that offer this capability, selecting the best one and then invoking it using a SOAP message over HTTP.



## Figure 1: Informal WSMX Architecture - high level overview

The two main objectives of the WSMX architecture are described in the next sections:

- Compilation of WSML messages describing aspects of web services
- Execution of goals passed to WSMX as SOAP message containing the goal as a WSML message.

### 2.1 Compilation

The user interface component provides the facility to create WSMO descriptions of aspects relating to web services. WSMO descriptions are specified in WSML and are passed to WSMX for compilation. Examples of WSML descriptions are available in the listings of the use-case deliverable 3.2 of the WSMO working group [[Stollberg et al., 2004]]. Compilation involves parsing the WSML message, verifying and validating the message against WSMO and then storing the parsed elements of the message in the WSMX repository. The repository can be either internal to WSMX or external as, for example, the extension to the UDDI described in the WSMO Registry [Herzog et al., 2004]. Once a WSML definition has been compiled to WSMX, it is available for use in the execution of user or application specified goals.

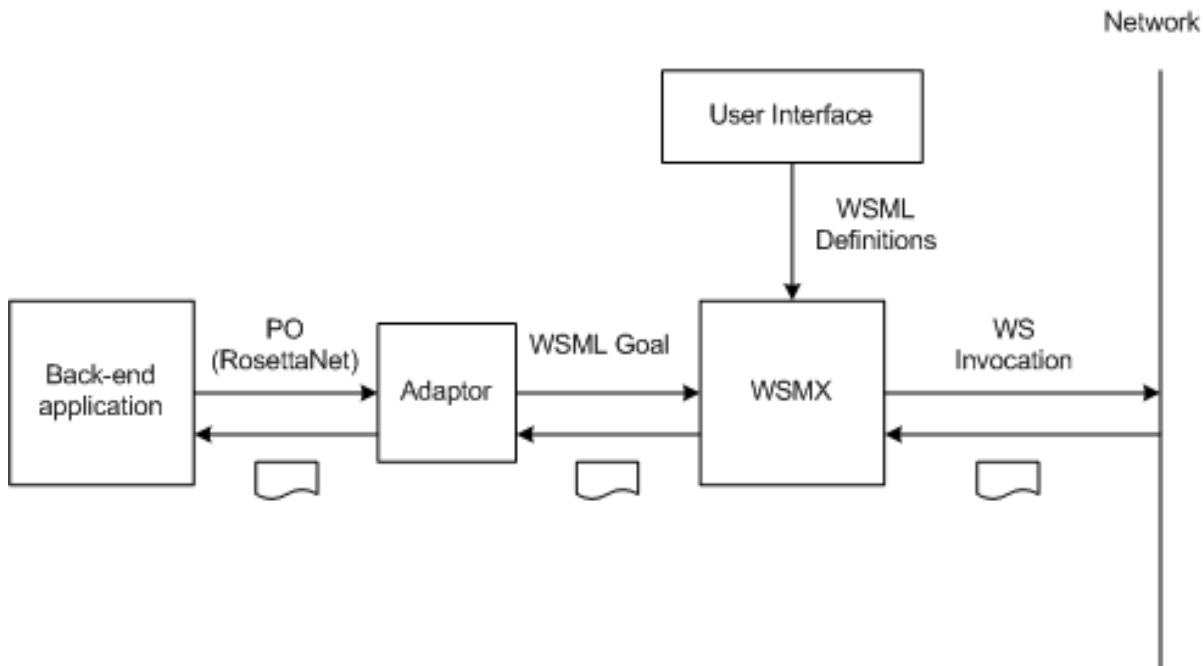
### 2.2 Execution Functionality

WSML Messages specifying goals that represent tasks to be carried out by WSMX are accepted by the WSMX Management component. An example of a WSML goal can also be seen in the WSMO Use Case deliverable 3.2. The WSMX Manager component coordinates and manages the process that is followed to achieve the goal. State information is stored at each stage and this information will, in time, be used to provide feedback on goal execution to a control module in the user interface. Typically the matchmaker component matches the goal to a capability known to WSMX and returns a collection of web services that offer this capability. The selection component selects the web service that provides the best match for the goal. How the selection component makes this decision is to be finalised. For the initial version of WSMX, it will be a simple algorithm. The mediation component finds a mediator in the WSMX repository that can mediate between the ontology used by the goal (source) and that used by the selected web service (target). Mediation is carried out based on a set of mapping rules between the two ontologies that are applied to the instance data contained in the goal. Once the data has been mediated, it is made available to the invoker which makes the call to the web service over the network.

## 3. Black Box Overview

The purpose of this section is to present the WSMX architecture from a simplified high level perspective to give an overview of what WSMX can do. This section provides an example of how external entities such as intra-enterprise back-end applications or inter-enterprise information systems can interact with the WSMX platform.

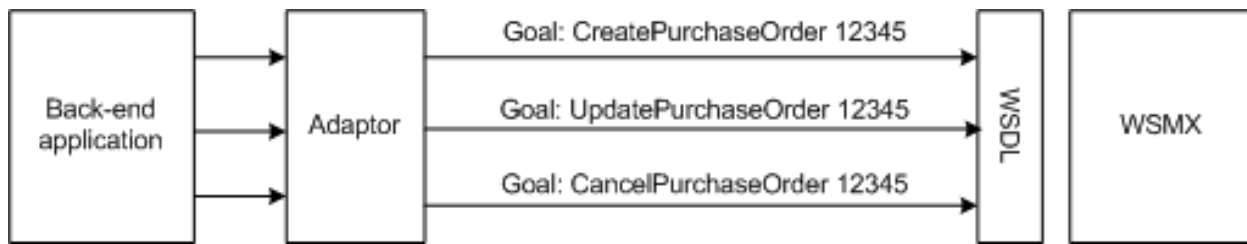
In the example below a back end application within an enterprise uses the RosettaNet [REF] B2B protocol to create messages required to purchase items of computer equipment. The application has the goal to buy computer equipment and has the condition that the messages it expects to send and receive with respect to this purchase will follow the RosettaNet protocol.



**Figure 2: WSMX External View**

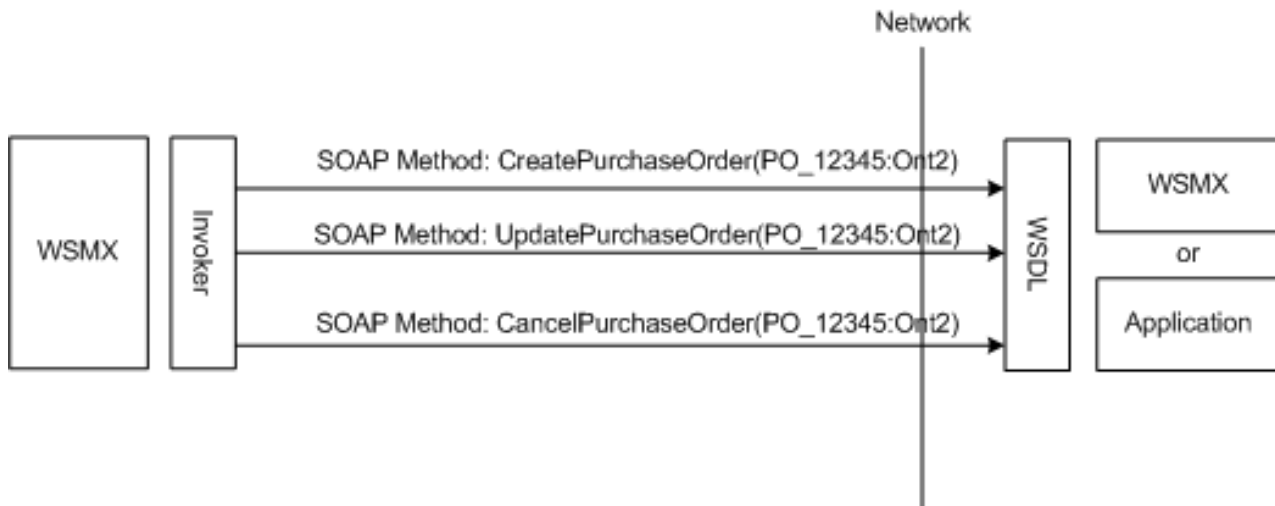
Figure 2 illustrates the simple example of a back end application sending a PO to trading partner. The application creates a RosettaNet purchase order (PO) for buying a laptop and sends this to the WSMX adaptor. The adaptor takes the information in the PO and generates a WSMX message consisting of a goal that describes what WSMX should execute and the information required by WSMX for this execution. Before WSMX can execute the goal, WSMO descriptions of the web services offering the capability to sell laptops, along with the WSMO description of the ontologies these web services use, and the RosettaNet ontology, must have been provided to WSMX. These descriptions are created using the user interface and then compiled to WSMX. When WSMX receives the WSMX message specifying the goal, it discovers the web service that best matches that goal, mediates the purchase order data following mapping rules between the RosettaNet ontology and the ontology of the selected web service and then invokes the web service providing the data to it in the concepts and formats it expects. The main point is that once the web services and ontologies are defined and compiled to WSMX, all the back end application has to do is pass the PO to the adaptor and WSMX takes care of the execution.

Figure 3 shows how WSMX itself is a web service and provides a WSDL interface that can be invoked either inside or outside an organisation's network. The diagram shows WSMX handling three operations relating to RosettaNet PIPs for purchase order handling.

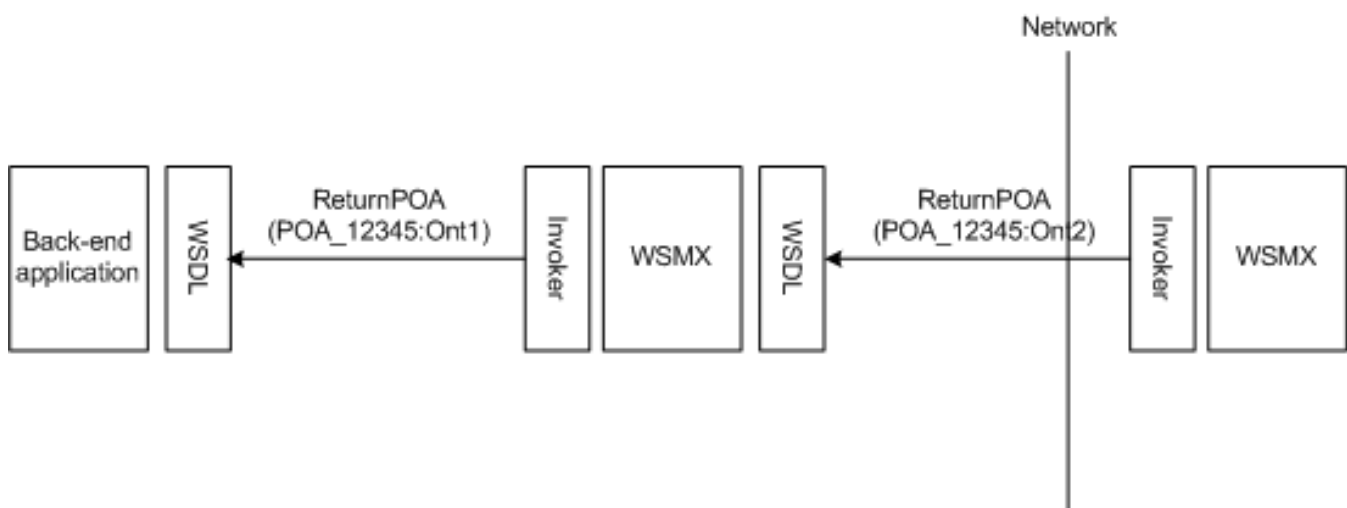


**Figure 3: WSMX as a Web Service**

Figure 4 shows WSMX communicating with an external web service while figure 5 illustrates how WSMX itself could be invoked as a web service from a WSMX in another organization.



**Figure 4: WSMX Invocation**



**Figure 5: WSMX to WSMX Communication**

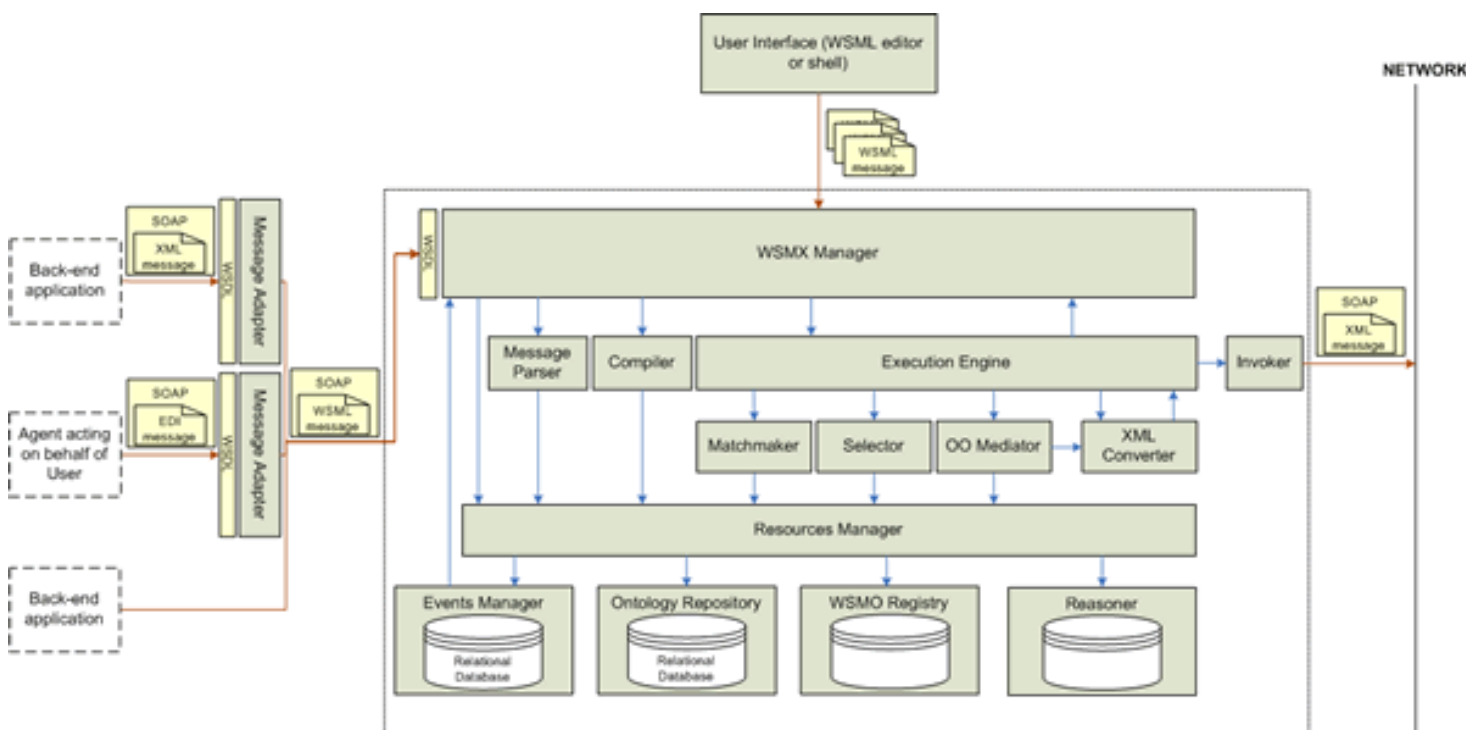
The WSMO descriptions of goals and web services provide the basis for the creation and execution of business processes in WSMX based on formal service descriptions rather than on human language and syntactical descriptions. Business process execution is outside the scope of the initial version of WSMX but will be based on the choreography and orchestration

deliverables of the WSMO working group.

## 4. Architecture Components and Interfaces

### 4.1 Architecture

WSMX architecture consists of set of loosely coupled components as presented on figure 6, which in most of the cases can be easily plugged-in and plugged-out from the system (e.g. OO Mediator provided with the reference implementation of WSMX can be easily substituted with the component provided by any commercial company). For each of these components public interface is defined, which can be either accessed by other components provided with the reference implementation, or by components provided by independent providers. WSMX reference implementation aims to be the complete implementation of all of the components, but users of the system may still decide to use components provided by other providers. This section aims to explain the functionality and the purpose of each of the WSMX architecture components.



**Figure 6: WSMX Architecture** ([click to enlarge](#))

There are two types of components in the WSMX architecture - components dedicated to compilation of WSML messages (messages, which describe **capabilities** of the Web Services), and components dedicated to run time execution of WSML messages (messages, which describe **goals** of requesters e.g. back-end applications or agents acting on behalf of user). Two of the components are specifically dedicated to the compilation process of WSML messages and eight of them are dedicated to the run time execution process of WSML messages. WSMX Manager and Resource Manager components are shared among compilation and execution functions of the WSMX platform. Components dedicated to compilation of WSML messages are: WSML Editor and Compiler. Components dedicated to run time execution of the WSML

messages are: Events Manager, Execution Engine, Message Parser, Matchmaker, Selector, OO Mediator, XML Converter and Invoker. Externally to architecture remains Message Adapters to allow transforming from any message format (e.g. RosettaNet, EDI, xCBL etc.) to WSMX message format. In the future WSMX aims to provide the framework, which will describe how to build new adapters and use them with WSMX. Both WSMX Manager and Message Adapters describe their interfaces using WSDL.

## 4.2. Components of WSMX Architecture

This section aims to define the functionality (scope) of components of the WSMX system as well as it defines interfaces for these components. Architecture is designed in a way to enable loose coupling of components. This section should be used as a reference by developers of the particular components.

### 4.2.1 User Interface - WSMX Editor

User interface component is used to create new WSMX definitions. User Interface (editor) initiates the compilation process of new WSMX definition into WSMX platform by sending SOAP messages with WSMX definition attachments. The graphical environment should enable owners of Web Services to describe capabilities of their Web Services using WSMX language. Any new definition can be saved in Ontology Repository (WSMO Registry). Particular implementation of editors will probably also provide the ability to store WSMX definitions locally on machines, which run WSMX editors. Saving WSMX definition remotely on the WSMX platform allows retrieving definitions, while working in distributed environment. Once the definition is complete it can be compiled into platform. Editor should support browsing of WSMX definitions compiled previously by a user and enabling removing any definitions, which should not be any longer supported (e.g. the Web Service can become obsolete and their definitions should not be longer available in WSMX). Access control mechanism based on roles, usernames and passwords should be implemented to enable restrictive access to some definitions and to reporting mechanism. WSMX aims to come with the set of predefined templates and a mechanism allowing creating new WSMX templates defined by users. Communication between User Interface component is based on a interface based on Web Services protocols (set of WSDL operations with their inputs and outputs), so both "fat" and "thin" clients editors written in any computer language can be provided to communicate with the WSMX Manager component.

This section described only a set of a suggested functionality, which should be provided by WSMX editors, but developers are advised to come with their own ideas to achieve competitive advantage of their own solutions. The detailed definition of WSMX Manager interface, which should be used by WSMX Editors, exposed as WSDL interface is now work in progress.

### 4.2.2 WSMX Manager

WSMX Manager is a gateway to WSMX. It can be accessed by two different WSDL interfaces. The first WSDL interface is an entry point for a run time WSMX messages arriving from agents and back-end application systems and the second WSDL interface is a gateway for WSMX definitions coming from WSMX Editors. WSMX manager processes business logic of the system

and observes/records any events happening on the WSMX platform.

Once requested by WSML Editor, WSMX manager schedules jobs to Compiler. If compilation is successful, WSMX manager as part of an output object, returns back "success" reply. Otherwise it returns "failure" reply back to WSML Editor. Reporting functionality is exposed through the WSDL interface as well.

WSMX Manager component provides **message listeners**, running as part of the application server, awaiting for any new messages incoming from the network. Each new message is handled by business logic implemented by the Manager component. WSML messages coming from the back-end application system or from the end-user of the platform are forwarded to Resource Manager to be permanently recorded in the datastore. Any message incoming from the WSML editor is given to the compiler. Based on the compilation result, the reply is given back to editor, and the objects defined in the WSML definition are recorded in the datastore as well. Independently from this, WSMX Manager handles business logic inside of the system. All events happening in the system are traced by this component and whenever new event happens or the status of the existing event changes, WSMX Manager takes care to make this information permanent.

WSDL interface exposed to back-end application systems is not a stand-alone component itself (because it does not provide any extra functionality), but it is only an endpoint which describes externally exposed functionality of the WSMX platform. This endpoint can be used by any external entities (e.g. back-end applications or information systems from other enterprises), which want to process WSML messages. There is only one external function/operation provided by WSDL interface to other information systems in WSMX platform called `receiveWSMLMessage` (see table 1).

Method Summary	
SOAPMessage	<p><code>receiveWSMLMessage(SOAPMessage message)()</code></p> <p>This operation takes as an input an instance of SOAP message, and returns back another instance of SOAP message. SOAP messages carries WSML payload in form of attachments (e.g. purchase orders), while output message would encapsulate WSML messages with unique identifiers UUID, which are assigned to new, instantiated events in the WSMX platform.</p>

**Table 1: WSMX Manager interface run time interface definition**

`receiveWSMLMessage` takes as input an instance of SOAP message, and returns back another SOAP message. While input messages would usually carry big attachments describing in WSML language the **goals** of requesting parties, they output messages returns back SOAP messages with unique identifiers assigned to a request.

### 4.2.3 Compiler

Compiler checks if the syntax of integration WSML definitions is correct. Only if correct,

definitions are decomposed into java objects as defined by class model and Compiler request Resource Manager to store them.

Method Summary	
Result	<code>compile(WSMLMessage message)()</code> Compiles WSMLMessage

**Table 2: Compiler method definiton**

#### 4.2.4 Message Parser

Message Parser provides functionality similar to the compiler, but it parses WSML messages processed at run time (messages coming from the back-end applications and from the end-users).

Method Summary	
Result	<code>parse(WSMLMessage message)()</code> Parses WSMLMessage

**Table 3: Parser interface method definiton**

#### 4.2.5 Resource Manager

Resource Manager is a gateway to any persistent storage used by a WSMX system.

#### 4.2.6 Ontology Repository

Ontology repository component is a persistent storage for integration definition types and for mediated instances of messages (it will be extended in the future to support selection and discovery components).

During run time, mediation component can use ontology repository to retrieve Mappings Rules, which are necessary to carry mediation.

#### 4.2.7 Events Manager

Events Management component saves events and their status, when they are happening in the system. An externally operations supported by this component and exposed by Resource Manager are presented in table 3 (this table is incomplete - will be updated in the next releases of this document).

Method Summary	

UUID	<code><u>saveMessage</u>(<u>SOAPMessage</u> message)()</code> Saves WSMX message
WSMXMessage	<code><u>getMessage</u>(<u>UUID</u> uuid)()</code> Retrieves WSMX message
UUID	<code><u>createNewEvent</u>(<u>EventName</u> name, <u>RelatedEvent</u> uuid)()</code> Creates new Event
Reply	<code><u>updateEvent</u>(<u>EventStatus</u> status, <u>EventUUID</u> uuid)()</code> Creates new Event

**Table 4: Events Manager interface methods definition**

## 4.2.8 WSMO Registry

A separate deliverable D10 defines WSMO Registry component, which is used by WSMX

## 4.2.9 Execution Engine

Execution engine controls the process of matchmaking, selecting, mediating and invoking of the Semantic Web Services.

## 4.2.10 Matchmaker

Matchmaker component provides the collection of Web Services, which are capable to meet given goal. Discovery match goals with the capabilities of Web Services.

Method Summary	
WebService[]	<code><u>match</u>(<u>WSMXGoal</u> goal)()</code> This method matches Web Services with goal specified by requester. Method receives WSMXGoal as an argument and returns back an array of Web Services.

**Table 5: Matchmaker interface method definition**

## 4.2.11 Selector

Selection component select only one Web Service from the collection of Web Services, which is capable to meet goal of the invoker, based on WSMX sender preferences.

Method Summary	

WebService	<pre><b>select</b>(WebServices[] webServices, Preference[] preferences) ()</pre> <p>Selects the best Web Service based on the provided preferences.</p>
------------	---

**Table 6: Selector interface method definition**

## 4.2.12 OO Mediator & XML Converter

Mediation engine is a Web Service so any new mediator can be plugged into WSMX instead of standard mediator provided with the platform. Mediator interface is presented in table 4.

Method Summary	
Payload	<pre><b>mediate</b>(String sourceOntologyID, String targetOntologyID, Payload message)()</pre> <p>This operation takes as an input identifiers of ontologies and a payload message, and returns back mediated Payload message</p>

**Table 7: Mediation Engine interface method definition**

## 4.2.13 Invoker

Invoker component calls Web Services in other information systems. Invoker converts objects into messages format, packs them into SOAP messages and invokes Web Service provided by other information systems.

# 5. Conclusions and Future Work

At this stage this document presents the general overview of the WSMX architecture and interfaces exposed by each of the components. Future work will include the detail specification of the functionality of each of this components, class model for objects of the system and data model used by the persistent storage.

## References

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

**[RosettaNet]** The RosettaNet consortium for open eBusiness standards and services, <http://www.RosettaNet.org>

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

**[Oren et al., 2004]** Oren et al., BNF grammar for WSML user language, <http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418>

**[Roman et al., 2004]** Roman et al., Web Service Modeling Ontology - Standard (WSMO - Standard), <http://www.wsmo.org/2004/d2/v02/20040306/>

**[Oren, 2004]** WSMX Execution Semantics, <http://www.wsmo.org/2004/d13/d13.2/v0.1/20040531index.pdf>

**[Stollberg et al., 2004]** WSMO Use Case Modeling and Testing, <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040524/>

**[Herzog et al., 2004]** WSMO Registry, <http://www.wsmo.org/2004/d10/v0.1/>

## 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](#)