



WSMX Deliverable  
D13.0 v0.1  
**OVERVIEW AND SCOPE OF  
WSMX**

WSMX Working Draft – 11th June 2004

**Authors:**

E. Oren  
M. Zaremba  
M. Moran

**Editors:**

E. Oren  
M. Zaremba  
M. Moran

**Reviewers:**

D. Roman

**This version:**

<http://www.wsmo.org/d13/d13.0/v0.1/20040611>

**Latest version:**

<http://www.wsmo.org/d13/d13.0/v0.1/>

**Previous version:**

<http://www.wsmo.org/d13/d13.0/v0.1/20040524>



# 1 Introduction

## 1.1 Overview

The Web Services Execution Environment (WSMX) is an execution environment for dynamic discovery, selection, mediation and invocation of semantic web services. WSMX is based on the Web Services Modelling Ontology (WSMO) [9] which describes all aspects related to this discovery, mediation, selection and invocation.

WSMX is a reference implementation for WSMO. The goal is to provide both a testbed for WSMO and to demonstrate the viability of using WSMO as a means to achieve dynamic inter-operation of semantic web services. The development process for WSMX includes defining its conceptual model, defining the execution semantics for the environment, describing an architecture and software design, and building a working implementation.

The term *semantic web services* refers to software components that use the Web Service Definition Language (WSDL) to define the syntax and data-types of their offered operations, and WSMO to formally define the functionality of those operations. Semantic web services are ordinary web services that are semantically annotated.

## 1.2 Purpose of this Document

This document provides an overview of WSMX in terms of its objectives, scope, development strategy and documentation. The purpose of the document is to explain the relationship between WSMO and WSMX, to describe the objectives of WSMX and to show how the development strategy provides a robust framework - allowing new developments in Semantic Web Service technology to be integrated with WSMX without affecting its stability.

## 1.3 Scope of WSMX

The final scope of WSMX is the domain defined by WSMO Standard. The scope of WSMX for the initial version is WSMO Lite [10]. Focusing on the restricted conceptual model provided by WSMO Lite allows the implementation of a simple extensible framework for executing Semantic Web Services. While the first version of WSMX provides a complete architecture for dynamic discovery, mediation, selection and invocation, the implementation of these components will be naive.

The initial functionality will allow achieving some user-specified goal by invoking a Semantic Web Service, providing it with a mediated ontology fragment. The first version of WSMX will be available at the end of June 2004.

Subsequent versions of WSMX will incorporate the ongoing research of the Semantic Web Services community, in particular the WSMO<sup>1</sup> working group and the WMSL<sup>2</sup> working group; augmenting and empowering WSMX to provide the

---

<sup>1</sup>see [www.wsmo.org](http://www.wsmo.org)

<sup>2</sup>see [www.wsmo.org/wsm1](http://www.wsmo.org/wsm1)



full Semantic Web Services support described in section 1.1.

## 1.4 Related Systems

WSMX is the first reference implementation of WSMO. However other implementations of execution environments for Semantic Web Services also exist. It is the intention of WSMX to be interoperable with such systems where possible. One such system is the Internet Reasoning Service (IRS-II) [7].

## 1.5 Document Overview

Section 2 describes the objectives of WSMX and differentiates between the initial and subsequent versions. Section 3 outlines the development strategy for WSMX and the rationale behind it. This section also provides two use cases illustrating how the framework implemented in the initial version of WSMX will be extended in subsequent versions. Section 4 describes the deliverables to be produced by the WSMX working group and explains how they relate to each other. Section 5 concludes the document.

## 2 Long-term functionality objectives

WSMX will offer complete support for interacting with Semantic Web Services. These web services must provide a WSMO-compliant description of themselves specifying their capability, how to interact with them, the ontologies they use along with other properties. As described above in the scope, the initial version will provide the framework for providing the functionality along with a naive implementation. Each subsequent version of WSMX will extend and improve the functionality of the components of the framework. The intended long-term functionality offered by WSMX includes:

**Services repository management** WSMX will offer management of a repository of available semantic web services. This repository could be centralised or decentralised. In [3] an enhanced UDDI registry is described, specifically designed to store, search, retrieve and manage WSMO descriptions of semantic web services<sup>3</sup>. This centralised repository could be populated either by the service providers (maintaining the semantic web services they offer) or by some web crawler, that finds available semantic web services and adds them to the repository.

The repository could also be decentralised, e.g. residing locally within the WSMX architecture; in this case it could be populated by the WSMX administrator adding the available semantic web services manually (using the WSMO editor) or by having a local web crawler searching for available semantic web services.

**Goal repository management** WSMX will offer management of a repository of specified goals using again either a centralised or decentralised repository, containing predefined goals, constructed by logical experts and domain experts. It should be possible for users to construct their goals, ideally with some tool

<sup>3</sup>this registry is currently being developed as part of the WSMX Working Group



support; another possibility would be to have users reuse (or refine) goals made by logical and domain experts.

**Service matching** WSMX will offer matching of usable semantic web services, returning either one web service with the capability to satisfy some specified goal, or returning a composition of semantic web services that together satisfy some specified goal.

**Service selection** WSMX will offer dynamic selection of discovered semantic web services based on specified preferences of nonfunctional criteria, including properties like quality of service, reliability, etc.

**Data mediation** WSMX will offer mediation of communicated data to be understandable by the selected web service

**Service invocation** WSMX will offer invocation of the selected semantic web service, obeying its prescribed choreography pattern. Here the issue of backwards compatibility comes into play, since most web services in the real world do not (yet) support WSMO. Those web services can only understand SOAP messages as defined in their WSDL. For these web services, adapters will be made, translating WSMX messages to SOAP messages as described in the WSDL of the invoked web service.

### 3 Short term functionality objectives

WSMX will follow the best practise of staged component-based software development. The initial phase will focus on constructing a rigid architecture for web service execution using decoupled, independent components. Once the WSMX framework is in place, subsequent phases will focus on refining individual components such as those for dynamic discovery and mediation. The rationale of this staged approach is to provide component-based software that is both maintainable and extensible.

By focusing on the architecture, we deliberately limit the initial functionality of WSMX and base it on WSMO Lite, a specific subset of WSMO targeted at easy implementation. It should be clearly noted that later versions of WSMX will address the full range of concepts from WSMO Standard.

Our goal for the initial version is to get a working framework. This framework will be designed in such a way that extending and improving functionality later is possible: the strong component decoupling will allow them to be replaced or extended to achieve richer functionality. The focus in the initial phase is on the architecture. The functionality offered by the initial version of WSMX is therefore a subset of the previously described functionality.

It is important to notice that because of an component based approach, changes can be implemented gradually over time. The interface of all components will be stable, but the implementation of those interfaces will improve over time. The other components however, do not have to be adjusted or rewritten to deal with these changes: from their viewpoint nothing changed, since the interfaces they can invoke are still the same.

The short-term functionality will differ from the described long-term functionality as follows:



**Repository management of services and goals** The repository of available semantic web services and of available goals are stored locally and will be managed manually.

There is a *discovery* component, but its functionality is limited to searching in the internal WSMX repository. The user has to manually fill this repository with available semantic web services. However, in a later version, we could easily extend this *discovery* component to use some centralised repository like a WSMO-UDDI, or to crawl the web searching for WSMO-enabled web services.

Again, because of the component-based architecture, we can easily extend functionality in later versions without affecting the behaviour and stability of the rest of the system. The other WSMX components only know that there exists a *discovery* component, that returns all available semantic web services. How this component works is not relevant to them, and can therefore easily be adjusted later.

**Service matching** The goal-capability matching will have a naive and trivial implementation: a match will be sought between the post-conditions and effects of the goal and the post-conditions and effects of some capability. This match will be a logical equality; this means that capabilities subsuming the goal, or capabilities subsumed by the goal will not be returned.

In WSMO Standard a goal specifies the objectives that a client may have when he consults a web service. A capability describes a relation between a certain state that has to exist, and a certain state that can be achieved by a web service. Now, if a user states his goal, the *goal-capability matching* component in the execution environment should use a logical reasoner to search for all semantic web services having a capability that satisfies this goal.

The reasoner should be able to find capabilities that exactly match this goal, as well as capabilities subsuming this goal (and possibly capabilities subsumed by this goal<sup>4</sup>), maybe even to suggest close matches.

However, since this is a logically difficult task, in the initial version of WSMX we do have this *goal-capability matching* component, but it is implemented in a very naive (and not very functional) way. As explained before, the goal-capability matching performed by this component is limited to exact matches of goals and capabilities, but does not consider subsumption relations.

Since the correct architecture is fully in place in the initial version of WSMX, subsequent versions of the execution environment will be able to provide full support for the goal definition from WSMO Standard. As WSMX is component-based, we will only have to change the internal behaviour of the *goal-capability matching* component to provide this extended functionality, not affecting the behaviour or stability of the rest of the system.

**Service selection** Dynamic selection of semantic web services should be based on some user preferences. However, in the initial version of WSMX the algorithm for this selection is not based on these preferences, but is achieved by simply picking the first matching web service.

**Data mediation** Mediation is not done by using some external mediator (as in WSMO), but by specifying mapping rules between ontologies. Mediation is achieved, if possible, by applying these mapping rules.

<sup>4</sup>if you are looking for a computer, and some web service offers second-hand computers, this capability is subsumed by your goal. This might be what you're looking for, depending on how you (as a user) define the task of finding a matching web service; it could however also be that you really want to have only a second-hand computer.



In WSMO Standard mediation is performed by some external mediator. To overcome differences in ontologies used by different semantic web services, different mediators can be used. In the initial implementation of WSMX mediation is performed by a *mediation* component, that uses predefined mapping rules to mediate between two ontologies. However, in later versions the much richer model of WSMO Standard, using different kinds of mediators, will be supported.

Again, the component-based architecture of WSMX is the key to achieving this. The *mediator* component can easily be changed to use some external mediators, instead of performing the mediation by itself.

**Service invocation** Since there is no choreography in WSMO Lite but only simple message exchange patterns, these predefined patterns are the only interaction styles supported.

## 4 WSMX Deliverables

As the development of WSMX is described in several deliverables that are strongly inter-related, we now briefly explain the different deliverables:

**Conceptual Model** The WSMX Conceptual Model [1] deliverable defines all concepts used in WSMX. This conceptual model is based on WSMO Lite, and extends it by introducing several concepts.

**Execution Semantics** The WSMX Execution Semantics [8] deliverable defines the operational behaviour of WSMX. This is a formal specification, describing exactly how the system behaves. The execution semantics ensures a common understanding among developers of what the system should do and how it should behave in all possible situations. The execution semantics also makes sure that different implementations of an execution environment for WSMO behave the same to the outside world (if they follow the same execution semantics). Thirdly, the formal nature of the execution semantics enables checking (and proofing) certain properties of the system.

**Mediation** The WSMX Mediation [5] deliverable describes how mediation is dealt with in WSMX, since mediation is not done by external mediators, but by the system itself. This deliverable describes several aspects related to this mediation, for instance how to define mapping rules, how a reasoner can do mediation out of those mapping rules, and introduces a tool that helps users write their mapping rules.

**Architecture** The WSMX Architecture deliverable [6] describes the architecture of WSMX. This architecture is the basis for the implementation. The deliverable explains the architectural decisions that were made, and describes the various components.

**Implementation** The WSMX Implementation deliverable [2] describes the implementation details of WSMX. It will discuss the technologies used as a platform for WSMX and the rationale for using them. The implementation deliverable will also provide the class and database model for WSMX as well as describing implementation issues not related to the WSMX architecture as described in [6].



**WSMX Editor** The WSMX Editor deliverable [4] describes the development of an editor for WSMO annotations of semantic web services. This editor will provide an environment where users can design semantic annotations for web services, goals, ontologies and mediators. This editor will provide support in this task.

**WSMO Registry** The WSMO Registry deliverable [3] describes a design an implementation of a UDDI-like registry for WSMO semantic web services. This registry allows storing, searching, retrieving and managing of WSMO descriptions of semantic web services.

## 5 Conclusion

Concluding, we design and build an execution environment for the dynamic discovery, selection, mediation and invocation of semantic web services. In the initial stage we focus on delivering a component-based architecture, ensuring a rigid foundation for later extension. We design all the components needed for a WSMO Standard execution environment, but limit their functionality to support only WSMO Lite. In a later stage these components can be (independently) extended to fully support WSMO Standard and to achieve dynamic composition of semantic web services.

## 6 Acknowledgements

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

The editor would like to thank to all the members of the WSMO working group for their advice and input into this document.

## References

- [1] E. Cimpian, A. Mocan, M. Moran, E. Oren, and M. Zaremba. WSMX Conceptual Model. WSMO Working Draft v0.1, 2004. Available from <http://www.wsmo.org/2004/d13/d13.1/v0.1/>.
- [2] E. Cimpian, A. Mocan, M. Moran, E. Oren, and M. Zaremba. WSMX Implementation. WSMO Working Draft v0.1, 2004. Available from <http://www.wsmo.org/2004/d13/d13.5/v0.1/>.
- [3] R. Herzog, H. Lausen, D. Roman, and P. Zugmann. WSMO Registry. WSMO Working Draft v0.1. Available from <http://www.wsmo.org/2004/d10/v0.1/>.
- [4] H. Lausen and M. Felderer. WSMO Editor. WSMO Working Draft v0.1. Available from <http://www.wsmo.org/2004/d9/v0.1/20040527/>.
- [5] A. Mocan, E. Oren, E. Cimpian, M. Moran, and M. Zaremba. WSMX Mediation. WSMO Working Draft v0.1, 2004. Available from <http://www.wsmo.org/2004/d13/d13.3/v0.1/>.



- 
- [6] M. Moran and M. Zaremba. WSMX Architecture. WSMO Working Draft v0.1, 2004. Available from <http://www.wsmo.org/2004/d13/d13.4/v0.1/>.
  - [7] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *2nd International Semantic Web Conference*, 2003.
  - [8] E. Oren. WSMX Execution Semantics. WSMO Working Draft v0.1, 2004. Available from <http://www.wsmo.org/2004/d13/d13.2/v0.1/>.
  - [9] D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology Standard. WSMO Working Draft v02, 2004. Available from <http://www.wsmo.org/2004/d2/v02/20040306/>.
  - [10] D. Roman, H. Lausen, E. Oren, and R. Lara. Web Service Modeling Ontology - Lite. WSMO Working Draft v0.1, 2004. Available from <http://www.wsmo.org/2004/d11/v0.1/20040405/>.