



WSMX Deliverable
D13.4 v0.2
WSMX ARCHITECTURE

WSMX Working Draft – 7th March 2005

Authors:

Michal Zaremba, Matthew Moran and Thomas Haselwanter

Editors:

Michal Zaremba

This version:

<http://www.wsmo.org/TR/d13/d13.4/v0.2/20050307>

Latest version:

<http://www.wsmo.org/TR/d13/d13.4/>

Previous version:

<http://www.wsmo.org/TR/d13/d13.4/v0.2/20050221>



Abstract

This deliverable describes Web Services Execution Environment Architecture - WSMX Architecture. The document provides both a high-level overview of the necessary system components and interactions between them (conceptual architecture) as well as a low-level definition of components interfaces, connectivity, applied events mechanism, etc.



Contents

1	Introduction	4
1.1	Rationale of the Architecture	4
1.2	Architectural View Structural, Behavioral and Physical	5
1.3	Conceptual versus Implementation Architecture	5
1.4	Document Overview	5
2	Architecture Background	5
2.1	Semantic Web Services	5
2.2	Conceptual Architecture for Semantic Web Services	5
2.3	WSMO, WSML and WSMX	5
2.4	Architecture Layering	5
3	WSMX Architecture	5
3.1	Architecture	6
3.2	Standardization WSMX External Behaviour	6
3.3	Dynamic Execution Semantics	8
3.4	Architecture Components	11
4	Architecture Components	11
4.1	Architecture Components	11
4.2	Interactions Between Components	11
5	WSMX to WSMX Communication	11
6	Critical Analysis and Outlook	11
7	Summary	11
8	Acknowledgement	11
9	Appendix B. Changelog	11



1 Introduction

In order to support distributed heterogeneous applications built by different vendors, Web Services specifications have been developed. Existing Web Services cornerstone technologies such as UDDI [?], WSDL [?] and SOAP [?] provide the basic functionality for discovering (UDDI), describing interfaces (WSDL) and exchanging messages (SOAP) in heterogeneous, autonomous and distributed systems.

While existing work on Web Services is a step in the right direction to improve interoperability of an existing systems, in practical terms available Web Services support operations which are limited to independent calls over a network, hard wired collaborations or predefined supply chains. The real world business interactions are much more complex than simple request-reply style interactions. Even if we can already envision building of a complex distributed supply chain, linking systems of many companies with the help of Web Services, these links between systems would be hardcoded involving enormous programmer support to get this virtual supply chain functional. Web Services specifications do not provide any mechanism to specify how to include any additional semantic information which would enable processing services without any human interaction.

To address existing limitations of Web Services, we provide a guideline and justification for an architecture for the Semantic Web Service (SWS) systems. Our work on the Web Services Execution Environment (WSMX) provides the guideline documentation and reference implementation for an execution environment enabling dynamic discovery, mediation, invocation and interoperation of Web Services. WSMX is based on the Web Services Modelling Ontology (WSMO) [2], an ontology for describing various aspects related to Semantic Web Services. A Web Service can be registered with WSMX by describing it in terms of WSMO, using the Web Service Modelling Language (WSML) [1], and then by invoking a registration interface provided by WSMX. System for the Semantic Web Services must enable binding service requesters with service providers using semantic information, without human interactions. Also all the data and process heterogeneity aspects must be addressed to enable fully automated execution of Web Services.

1.1 Rationale of the Architecture

This document describes WSMX architecture. The rationale behind the architecture clarify design decisions and simplify in in understanding architecture and its goals:

Components decoupling Principle of SOA and WSMO about strong decoupling of the various components that realize an e-commerce application is also a rationale behind WSMX architecture. Making components self-contained supports a clear separation of concerns. Each component has a well defined functionality that other components can use.

Standardization of external behaviour The goal of this architecture document is to standardize the external interface and behaviour of components and the whole system. At this stage of work on architecture we assume that only components conforming to interfaces described in this documentation can be used in the architecture. As in the future we would like Semantic Web Services systems to be capable to support dynamic execution semantics, we also plan to design a mechanism to incorporate components with arbitrary interfaces, not standardized in this document.



1.2 Architectural View Structural, Behavioral and Physical

1.3 Conceptual versus Implementation Architecture

The goal of this deliverable is to provide both a high level conceptual architecture with an overview of system components and their functionality, as well as to focus on more technical aspects of the system for the Semantic Web Services execution. Although some technical aspects of the reference implementation system are discussed, this document does not aim to be a technical system documentation nor a programmers reference. All detailed technical aspects are going to be documented together with reference implementation of the system. This document is of a more philosophical nature, attempting to capture the thoughts and intentions for generic system for Semantic Web Services system, as well as the major factors driving design of the real system.

While our overall target is to build a distributed architecture for the Semantic Web Services, design details focus on WSMX reference implementation aspect providing a node-centric architecture of a WSMO execution environment.

1.4 Document Overview

2 Architecture Background

2.1 Semantic Web Services

2.2 Conceptual Architecture for Semantic Web Services

2.3 WSMO, WSML and WSMX

2.4 Architecture Layering

3 WSMX Architecture

WSMX is a Service Oriented Architecture (SOA), which means that it is a software system consisting of a set of collaborating software components with well defined interfaces that together perform a task. These components do not necessarily live in the same address space, not even in different address spaces on the same machine, instead they may very well live on different continents communicating over a network channel through multiple protocol stacks. This situation creates its own unique demands, born out of latency, memory access, concurrency and failure of subsystems, which the architecture must be able to cope with. All these aspects are going to be addressed in subsequent steps during designing and implementing reference implementation of WSMX. SOAs differentiate themselves from other distributed systems through the concept of loose coupling brought to its extremes. Strong de-coupling of the various components that realize an e-commerce application is one of major feature of WSMO. In WSMX conceptually independent pieces of functionality has been grouped in components. Each of WSMX components provides services - a single of which is a logical unit of system code, application code, persistency layers, in short anything that as a unit can carry out an application-level operation. Services are often characterized by exactly these operations, which they provide to



other components. Preferably services have descriptions in machine-processable meta-data. Although in this document we recommend to build communication of Semantic Web Services system on event paradigm, one can envision infrastructure coordinated without events. In distributed SOA system, communication between components is taking place through events. It enables the good practice of implementation-hiding, in which the implementation of a service should be of no concern to the client of the service. All of this together should result in increased flexibility, better extensibility and dramatically improved reusability. However it is not always easy to achieve all of these goals simultaneously, even if the right architectural decisions were taken. Scalability and proper service structuring are crucial and have to be taken into account.

3.1 Architecture

WSMX architecture consists of set of loosely coupled components as presented on figure 1, which in most of the cases can be plugged-in and plugged-out from the system (components provided with reference implementation of the system can be easily substituted with other component provided by third parties). For each of these components public interfaces are defined, which can be either accessed by other components provided with the reference implementation, or by components provided by independent providers. The 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.

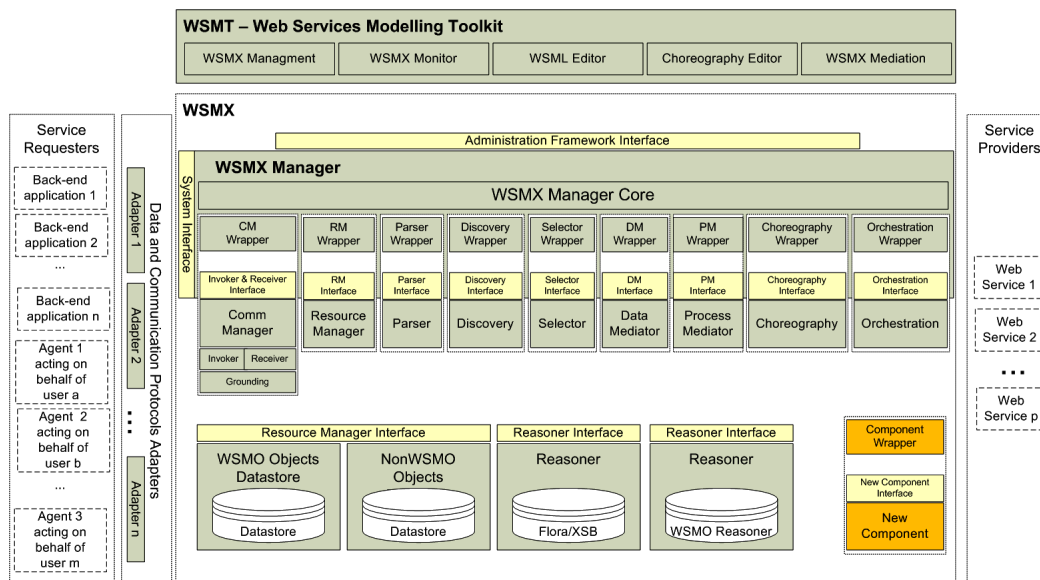


Figure 1: WSMX Architecture

3.2 Standardization WSMX External Behaviour

Functionality provided by WSMX system as the whole can be described in terms of its entry points - the standardized interfaces of the system enabling communication with any external entities requesting services from the system.



More details on system functionality can be found in an execution semantics document, which formally specify the desired operational behaviour of WSMX, serving not only as a reference for developers but also as a means of validation and model-checking.

In the current version of the system we define execution semantics with four possible branches each of them starting with one entry point. These four mandatory entry points must be available in each working instance of system, which is WSMX complainant. Entry points also define the required functionality of WSMX compliant system. By selecting given entry point the predefined execution semantics of a given set of components is triggered. These four obligatory entry points enabling execution of any of four available execution semantics are:

realiseGoal(Goal, OntologyInstance):Confirmation

Any external entity, which expects to get its goal realised without back and forth interactions (communication) with WSMX/DIP system, might wish to provide a formal description of a Goal (in WSMO terms) and instance of Ontology. This quite simplified scenario assumes that service requester knows even before service discovery all the data, which might be required by service provider. WSMX/DIP selects and executes service on behalf of service requester. Service requester might receive final Confirmation, but this step is not obligatory (many entities, which might wish their goals to be realised by WSMX/DIP system might not have permanent addressing, so there is no possibility to make an asynchronous call back to them returning the final result of the service invocation).

receiveGoal(Goal):WebService[]

ReceiveGoal entry point addresses more realistic scenario, when service requester might wish to consult WSMX to learn about Web Services which satisfy its Goal. In this asynchronous call, service requester provides Goal and expects to get back a set of Web Services.

receiveMessage(OntologyInstance, WebServiceID, ChoreographyID):Confirmation

Once service requester knows Web Service which he wants to use, he must carry back and forth conversation with WSMX/DIP system to provide all the necessary data to make execution of this Web Service feasible. By giving fragments of Ontology Instances (e.g. business documents such as Catalogue Items or Purchase Orders in a given ontology) and reference to WebService and Choreography (only if choreography has been instantiated already), which should be used, it provides all data required by Web Service of service provider.

storeEntity(WSMOEntity):Confirmation

Store Entity entry points provides an administration interface for the system enabling to store any WSMO related entities (like Web Services, Goals, Ontologies) and making them available for other parties using WSMX/DIP system.

Additionally to these four entry points, we assume that WSMX/DIP provides an engine to support dynamic execution semantics enabling execution of any formal description of system behaviour. In this way we can also define additional and not required by any implementation, functionality of the system.



Four entry points described in this section are fundamental to describe sequence diagrams, which we will present in later parts of this document (at this stage we have focused on providing sequence diagrams for the second and third option, as well as we have started already to work on addressing the first option as well).

3.3 Dynamic Execution Semantics

This section present some initial thoughts on dynamic execution semantics in WSMX. It is still very first draft and very high level overview of architecture enabling dynamic execution semantics.

During design process of WSMX several steps have been undertaken, including describing its conceptual model, specifying its formal system execution semantics and designing its architecture. By providing the conceptual model the common reference vocabulary for the development team has been defined, which has been meant to be used at different phases of the project. Execution semantics - the formal specification of the operational behavior is normally used for a number of reasons during software development. As described by [4] in the context of WSMX we were initially interested in modeling the execution semantics of WSMX in such a way that developers understand the system, that certain properties of the system can be inferred and that it enables model-driven execution of the system's components. Architecture definition has delivered a detailed description of the overall system, components interfaces and specification of the functionality expected from the particular components. One of the key design decisions we undertake for WSMX has been to keep individual components decoupled one from each other and to enable components distribution across the network. The development of the high-speed networks makes it possible to distribute services across various machines achieving complete functionality by combining these partitioned and distributed resources. The effective way to exploit advantages provided by network and achieve system scalability requires partition system functionality into coarse-grained components with well defined interfaces which can provide a small but complete functionality required by this system. An initial version of WSMX included only one possible execution semantic which was hard-coded into components of the system. This approach very quickly showed weaknesses of WSMX design as revealed by new requirements and use cases provided by potential users of the system. We decided to refine our initial architecture approach and we accommodated a mechanism to incorporate new components and methods enabling adding and removing any new formal definitions of execution semantics describing operational behavior of the system without the need to recompile the whole platform any time such new execution semantic becomes available. The first step to enable new execution semantics in the system has been the design of execution semantics (or in SOA terminology - "business processes") - a group of business activities undertaken by a management component in pursuit of a common goal. While for WSMX we used Petri nets [3], we do not restrict dynamic execution engine to any specific formalism. Tool we are using, CPNTools, make it possible to model so-called high-level Petri-nets, extending classical Petri nets with hierarchy, color and time. The tool used for definition of business process must make it possible to verify certain properties of the model; it must check some simple properties such as syntactical correctness, unreachable states or unsatisfiable conditions. While we define process, we made an abstract declarations that services will be requested by using services interfaces (see figure 2), but actually we do not bind the process to the concrete service, which is going to be invoked during



run time. As an underlying formalism for defining abstract business processes in WSMX dynamic semantic execution engine we have started using YAWL, a novel workflow management language. It builds on the formal foundations of Petri nets but is specifically designed for usability, which could be an advantage over a purely Petri net based approach. The system includes an enactment engine and a design tool; the system is however quite young and not yet mature. We have already made some initial tests in using YAWL as system behavior definition formalism for WSMX.

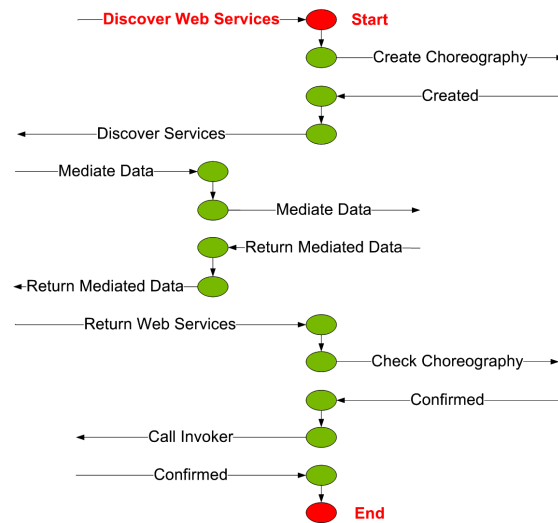


Figure 2: Abstract Execution Semantics Definition

Except process definition, we recognized another requirement to enable dynamic execution semantics - runtime ability to plug-in and plug-out components. In WSMX we enable reconfiguration, management, and monitoring of available software components. We maintain that system must be capable to host deployable components and to reconfigure them during initialization and as during runtime (see listing 1 for system configuration file definition).

`<XML configuration file will be pasted in here>`

The persistent configuration support is responsible for loading the various components into memory at startup time. Different versions of WSMX provide different degrees of configuration support, early releases provide only basic, centralized support while later releases are planned to have more sophisticated and decentralized configuration systems to provide more flexibility. The system must be able to cope with the additional complexity caused by the introduction of features such as component injection or persistence of metadata objects. Having abstract process definition and components installed in the system, we generate wrappers for components (see figure 3). The purpose of the wrapper is to separate components from transport layer for events. As mentioned before although we also recommend building the communication inside Semantic Web Services architecture based on event paradigm, the system components can be coordinated without events. WSMX is an event-based system, consisting of many wrappers that communicate using events. Wrappers exhibit an asynchronous form of communication. One wrapper raises an event with some message content and another wrapper can at some point in time consume this event and react upon it. Components offering services to WSMX remain unaware of event infrastructure, while they solely communicate with their own wrappers, while event consumptions and production is taking place only on a wrapper



level. Transport mechanism has been also decoupled from the system by using Transport interface, which hides details of event transport mechanism.

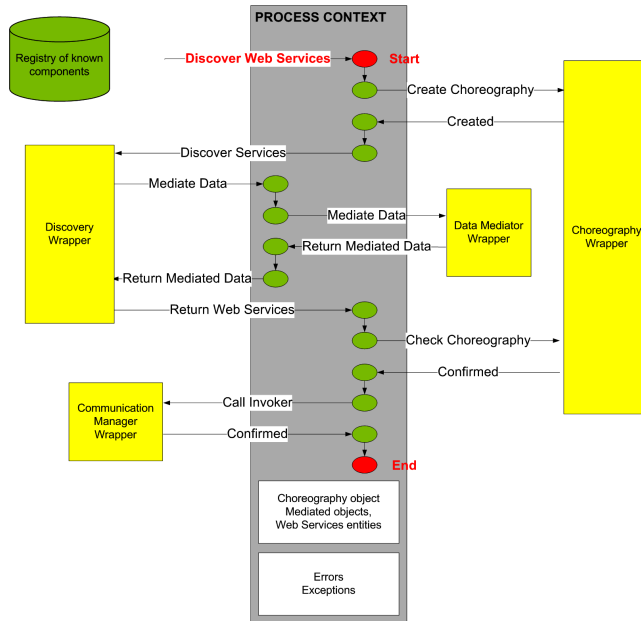


Figure 3: Process, its wrappers and context

Figure 4 presents the complete architecture enabling decoupling of components from execution semantics in Semantic Web Services architecture. Deployment of any new execution semantics will regenerate wrappers for the set of components. Based on execution semantics definition, these wrappers will be only capable to consume and produce particular types of events. In a running system dynamic execution semantics is achieved by mapping abstract system behavior into real event infrastructure of the system.

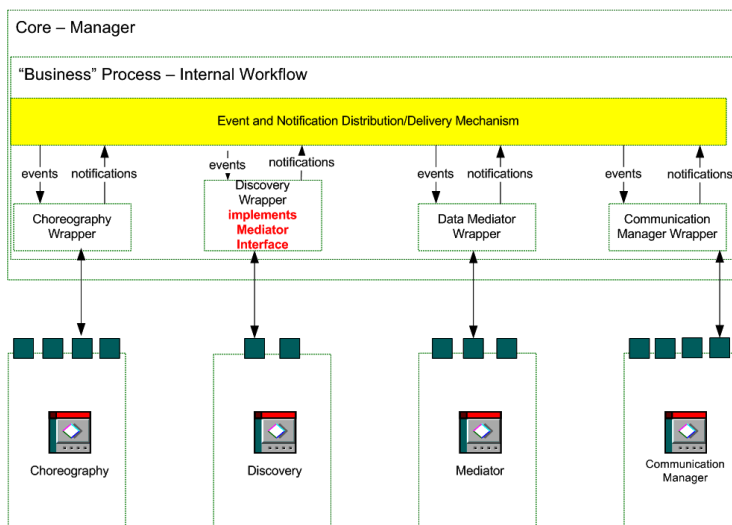


Figure 4: Event SOA for WSMX

To assure that there are no design flaws, livelocks or deadlocks the formal verification of the model must take place before model is deployed to the system.



3.4 Architecture Components

4 Architecture Components

4.1 Architecture Components

4.2 Interactions Between Components

5 WSMX to WSMX Communication

6 Critical Analysis and Outlook

7 Summary

References

- [1] Jos de Bruijn. D16 The WSML Specification. Technical report, WSML Working Draft, 2004. Available from <http://www.wsmo.org/2004/d16/v0.2/20041029/>.
- [2] D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology Standard. WSMO Working Draft v02, 2004.
- [3] W. M. P. van der Aalst, K. M. van Hee, and G. J. Houben. Modelling workflow management systems with high-level Petri nets. In G. de Michelis, C. Ellis, and G. Memmi, editors, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50, 1994.
- [4] J. M. Wing. A specifier’s introduction to formal methods. *IEEE Computer*, 23(9):8–26, September 1990.

8 Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperanto; by Science Foundation Ireland under the DERI-Lion project; by the Vienna city government under the CoOperate programme and by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW² and TSC.

The editors would like to thank to all the members of the WSMO, WSML, and WSMX working groups for their advice and input into this document.

9 Appendix B. Changelog

2005.03.07

- Added first cut for dynamic execution semantics