



WSMX Deliverable
D13.0 v0.2
**OVERVIEW AND SCOPE OF
WSMX**

WSMX Working Draft – 4th February 2005

Authors:

Emilia Cimpian
Matthew Moran
Eyal Oren
Tomas Vitvar
Michal Zaremba

Editors:

Emilia Cimpian
Tomas Vitvar
Michal Zaremba

Reviewers:

This version:

<http://www.wsmo.org/TR/d13/d13.0/v0.2/20050204>

Latest version:

<http://www.wsmo.org/2004/d13/d13.0/>

Previous version:

<http://www.wsmo.org/2004/d13/d13.0/v0.2/20050201>



Abstract

This deliverable is intended to be a high-level overview document of the research carried out by the Web Services Execution Environment (WSMX) working group. WSMX is an execution environment enabling discovery, selection, mediation, invocation and interoperation of the Semantic Web Services (SWS). The development process for WSMX includes establishing its conceptual model, defining its execution semantics, developing the architecture, designing the software and building a working implementation of the system. Through our research on WSMX we aim to provide a guideline and justification for an architecture for the SWS systems. In our deliverables we present how SWS architecture should look like, we define its execution semantics, mediation, discovery and invocation mechanisms, and any other aspects we perceive compulsory in any SWS related systems. Work carried out by WSMX working group except its research focus provides also a reference implementation of the Web Services Modeling Ontology (WSMO)[18]. The mission and ultimate goal of the working group is to define SWS architecture and build a full fledged enterprise application based on the conceptual model of WSMO.

Except the cornerstone SWS functionalities that should be available with any execution environment for the SWS such as discovery, selection, mediation, invocation and interoperation, also some more specific enterprise systems functionalities are getting addressed while developing WSMX system (although they remain out of scope of WSMO, as WSMO is only concerned with the external behavior of the system). Some of these additional functionalities, which are going to be available with the subsequent WSMX system implementations include (1) plug-ability mechanism for any distributed component unknown prior design of the system, (2) internal workflow engine capable to execute formal descriptions of behavior of the system components and (3) resource manager enabling persistence of any arbitrary data (WSMO and non-WSMO related) produced by components during run time of the WSMX server. Along with the development of the system, all important aspects of the enterprise software are going to be addressed as well. That is why in the subsequent releases of the system such aspects like reliability, scalability, security, etc. are going to be addressed and become an integral part of the WSMX server.

In this document, we briefly overview research carried out by WSMX working group, expected functionality of WSMX system as the whole and functionality provided by some of its required components, we present the current status of the development and we demonstrate WSMX specific deliverables produced by WSMX working group.



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 [2], WSDL [3] and SOAP [13] provide the basic functionality for discovering (UDDI), describing interfaces (WSDL) and exchanging messages (SOAP) in heterogeneous, autonomous and distributed systems. In practical terms, existing Web Services support operations which are limited to independent calls over a network, hard wired collaborations or predefined supply chains. Web Services specifications do not provide any mechanism to specify how to include any additional semantic information which would enable processing them without any human interaction.

The approach to systems integration based on semantically enhanced Web Services is nowadays achievable through SWS. Web Services Modelling Ontology (WSMO) initiative¹ is one of the several efforts around the world working on providing conceptual model, language and execution environment for SWS. Enhancing existing Web Services standards with semantics markup standardized through WSMO initiative promotes already existing Web Services standards for semantic-enabled integration.

The Web Services Execution Environment (WSMX)² is one of the WSMO initiatives aiming at providing an execution environment for discovery, selection, mediation, invocation and interoperation of the SWS. The aim of research on WSMX is to provide a guideline and justification for an architecture for the Semantic Web Service (SWS) systems. WSMX is based on the conceptual model provided by Web Services Modelling Ontology (WSMO) [18] which describes all aspects related to SWS. WSMX is also a reference implementation for WSMO being a vehicle for driving new projects and partnerships. The goal is to provide both a testbed for WSMO and to demonstrate the viability of using WSMO as a means to achieve dynamic interoperation of SWS. The development process for WSMX includes defining its conceptual model (which has been currently completely aligned with concepts provided by WSMO ontology), modeling its execution semantics (we already consider including dynamic execution semantic engine capable of interpreting any formal definition of system behavior provided by third parties) and collaborating on designing of system architecture. WSMX working group also defines components interfaces (this is also subjected to changes in the future as we already envision scenarios where providers of the components would like to provide their own interfaces for their own components and still to be able to connect them into WSMX core), designs particular components and provides their implementation.

This document provides an overview of WSMX in terms of scope, functional objectives, development approach and deliverables in relationship with WSMO.

1.1 Scope of WSMX Development

WSMX is the reference implementation of WSMO, thus its final scope will be in line with WSMO. The initial version of WSMX has been based on a refined and minimal set of WSMO concepts, while subsequent versions takes into consideration the complete conceptual model provided by WSMO. Already for the initial version of WSMX, a complete architecture including discovery, mediation, selection and invocation components has been designed including also all the supporting components enabling exchanging of messages between mock-up service requesters and providers. At early stages of WSMX development, the

¹Web Services Modelling Ontology working group: <http://www.wsmo.org/>

²Web Services Execution Environment working group: <http://www.wsmx.org/>



implementation of these components has been simple, while with each subsequent implementation new and more complex aspects are going to be addressed. WSMX initial functionality allows achieving a user-specified goal by invoking Web Service described with the semantic markup. The first version of WSMX has been available from June 2004 at the sourceforge portal³.

Subsequent versions of WSMX have been incorporating the ongoing research of the SWS community, in particular the WSMO working group and the WSML⁴ working group; augmenting and empowering WSMX to provide the full SWS support.

1.2 Development Approach

WSMX follows the best practise of staged component-based software development. *Initial phase* was focused on constructing a rigid architecture for Web Service execution using decoupled, independent components. Having already basic WSMX framework in place, *subsequent phases* we are focusing on refining the individual components, the whole framework and on providing new aspects. The rationale of this staged approach is to provide component-based software that is both maintainable and extensible. It is important to notice that because of an component based approach, changes can be implemented gradually over time.

1.3 Document Overview

Section 2 of this deliverable describes the objectives of WSMX in terms of the main functionality it should provide. Section 3 outlines the development approach for WSMX. Section 4 describes WSMX functionality delivered through its initial development phase in contrast with the main functionality objectives. Section 5 describes the deliverables to be produced by the WSMX working group and explains how they relate to each other. Section 6 compares WSMX to other similar systems. Section 7 concludes the document.

2 Expected WSMX Functionality

WSMX in its full potential will offer complete support for SWS. Through our research on WSMX we aim to provide a guideline and justification for an architecture for the SWS systems. WSMX deliverables present how SWS architecture should look like, define its execution semantics, mediation, discovery and invocation mechanisms, and any other aspects compulsory in any SWS related systems. The ultimate goal of the working group is to define SWS architecture and build a full fledged enterprize application based on the conceptual model of WSMO. Expected functionality of WSMX can be described in terms of aggregated functionality of all its components.

At this stage following main components has been already envisioned for WSMX:

Core Component Core Component is a heart beating component of the system, when all the other components are going to be connected into it. Any of

³WSMX at sourceforge: <http://sourceforge.net/projects/wsmx>

⁴Web Services Modelling Language working group www.wsmo.org/wsml



the interactions between components is coordinated through core component. Business logic of the system, events engine, internal workflow engine, distributed components loading etc. are all going to be subcomponents of the Core Component. While at this stage we still consider Core Component to be a central module for the system, in the future to ensure reliability of WSMX, the clustering mechanism for distributed Core Components might be also developed. For resources intensive operations instead of having only one instance of Core Component, many instances could be instantiated on several machines to create cluster of WSMX.

Resource Manager Resource Manager is responsible for management of repositories to store definitions of any WSMO (Web Services, goals, ontologies and mediators) and non-WSMO related objects for WSMX. Depending on the scope of information stored in the underlying registries, we distinguish registries to be local and global. Information stored in the local registry is relevant for domain-specific operations (e.g. most of the system run time information will be available only to local instance of WSMX) whereas global registry could be shared across several domains (e.g. registries of SWS descriptions). While both stored data and accessibility to this data might considerably differ between registries, the Resource Manager remain the only entry point for them (it is not possible to access any of registry directly, but only through Resource Manager).

Web Services Repository deals with semantic description of Web Services, such as their capabilities (pre-conditions, post-conditions, assumptions and effects), interfaces (choreography and orchestration) and non-functional properties including both general as well as Web Service specific. All information stored in Web Services Repository is related to case scenarios in which this information is to be used, such as discovery of services or monitoring of services.

Goals repository deals with semantic description of general goals. A WSMO Goal in most cases expresses the "wish" that a user wants to achieve. Some of goals may be reusable and provided as templates and therefore should be published using the repository. Non reusable or frequently updated goals should be stored at the service requester side or other locations. The goal repository can thus contain predefined goals constructed by domain experts as well as user-specific goals. Tool to construct/refine goals will be created for this purpose.

Ontology repository deals with ontologies to be stored in the registry describing semantics of particular domains. Any components might wish to consult ontology, but in most of the cases ontologies will be used by mediator related components to overcome data and process heterogeneity problems. We recognize the requirement for visualization tool and management mechanism for ontologies, that should be provided by WSMX. Ontologies stored by WSMX are expressed in terms of WSML, but we consider also the case that the syntactical adaptation from any ontology syntax is possible, before storing them.

Mediator repository deals with mediators to be stored in the registry. WSMO mediators has become a standard proposal covering various approaches for data and process mediation. While mediators in WSMO are still underspecified, we already consider storing them in repository for possible use of the data and process mediation components.

Data repository During run time of the system lots of system specific data is produced. Also components might generated data, which should be persistent. Data repository allows us to store any system data and any component specific data required for correct system execution.

Service Discovery The main purpose of service discovery is to provide functionality on matching of usable SWS with the goals. When matching SWS, a



number of them could be selected which capabilities satisfying criteria specified by a goal. A number of SWS could be returned from this step.

Service Selection Selection of the Web Service might happen in WSMX and for that purpose selection component is used. Selection can also take place outside of WSMX after service requester is presented with the set of potential Web Services satisfying his goal. When selecting Web Services, one, possibly "best" or "optimal" variant of a service should be returned from a set of satisfying matching criterions. To select an optimal service, different techniques can be applied, ranging from simple selection criteria ("always the first") to more sophisticated techniques, such as multi-criteria selection of variants also involving interactions with a service requester. Different variants of services could be described by different values of parameters (non-functional properties specific to web services), such as financial properties, reliability, security, etc.

Data and Process Mediator Mediation is needed when two entities that cannot communicate directly need to interact. The reason why they cannot communicate is that they are using different syntax or semantics of data they are dealing with and/or process for the same operation might mismatch. WSMX offers mediation of communicated data and used processes to be understandable and useful for the selected Web Services. Data mediation is based on paradigms of ontology management, ontology mapping and aligning in particular.

Communication Manager WSMX will offer invocation of selected services, obeying their interfaces, that is choreography and orchestration patterns. Communication Manager consists of two subcomponents: invoker and receiver. WSMX is an intermediary system sitting in between service requester and provider, so communication in both ways is supported by these two subcomponents. In general, invocation of services could be based on any underlying protocol of service providers (e.g. SOAP, proprietary protocols, etc.) and for this purpose adapter framework is designed. It would be the responsibility of an adapter to implement interactions with a particular service providers using communication protocols different than SOAP.

Choreography The choreography of a Web Service defines its communication pattern, that is, the way a requester can interact with it. 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. Since the clients communication pattern is in general 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. The role of the Choreography Engine is to mediate between the the requester's and the providers's communication patterns. This means to provide the necessary means for a runtime analysis of two given choreography instances and to use 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.

WSMT Web Service Modeling Toolkit (WSMT) is a framework for the rapid deployment of graphical administrative tools, which can be used with WSMX



Reasoner Although development of a reasoner is not part of the WSMX development process, WSMML compliant reasoner will be an important element of the whole WSMX framework. Reasoner will provide reasoning services for mapping process of mediation as well as functionality relating to validation of a possible composition of services, or determination if a composed services in a process is executable in a given context. Also, reasoner will be used for finding capabilities that exactly match requester's goal as well as capabilities subsuming this goal.

Except the cornerstone SWS functionalities that should be available with any execution environment for the SWS such as discovery, selection, mediation, invocation and interoperation, also some more specific enterprise systems functionalities are getting addressed while developing WSMX system. Some of these additional functionalities, which are going to be available with the subsequent WSMX system implementations include (1) plug-ability mechanism for any distributed component unknown prior design of the system, (2) internal workflow engine capable to execute formal descriptions of behavior of the system components and (3) resource manager enabling persistence of any arbitrary data (WSMO and non-WSMO related) produced by components during run time of the WSMX server. Along with the development of the system, all important aspects of the enterprise software are going to be addressed as well. That is why in the subsequent releases of the system such aspects like reliability, scalability, security, etc. are going to be addressed and become an integral part of the WSMX server.

3 Currently Addressed WSMX Functionality

We deliberately limited the initial functionality of WSMX by providing only necessary components to run end-to-end communication. It should be clearly noted that later versions of WSMX will address the complete set of the functionality required to execute SWS.

Our goal for the initial version has been to get a working framework. This framework is 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 has been on the designing and implementing a flexible architecture.

Core Component A heart beating component of the system has been provided already with the first release. Multi-threaded approach for components connectivity has been adapted. JMX technology has been used for components management. The execution semantics has been hardcoded into this component.

Resource Manager The repository of available SWS and of available goals is stored locally. There is a *discovery* component, but its functionality is limited to searching in the internal WSMX repository (Resource Manager remains localized). User has to manually fill this repository with available SWS. Over time the development of Resource Manager and its repositories has been moved from rigid JDBC⁵ technologies with MySQL⁶ database as an underneath storage, through EJB⁷ framework towards more lightweight framework based on

⁵Java Database Connectivity: <http://jcp.org/en/jsr/detail?id=054>

⁶MySQL database: <http://www.mysql.com/>

⁷Enterprise Java Beans: <http://www.jcp.org/aboutJava/communityprocess/edr/jsr220/>



Hibernate and Spring⁸.

Service Discovery The goal-capability matching has a simple implementation: a match is sought between the post-conditions and effects of the goal and the post-conditions and effects of some capability. This match is a logical equality; this means that capabilities subsuming the goal, or capabilities subsumed by the goal will not be returned. There is already a simply key-word matching discovery implemented.

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 SWS 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, 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 simple way.

Service Selection In the initial version, the algorithm for selection is achieved by simply picking the first matching web service. However, user preferences and expectations as well as non-functional properties of services should be involved in the selection process. We also consider another scenario where service is selected outside WSMX server by service requester and the selected Web Service is already given to the system.

Data and Process Mediator 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.

In WSMO Standard mediation is performed by some external mediator. To overcome differences in ontologies used by different SWS, different mediators can be used. In the initial implementation, WSMX mediation is performed by a *mediation* component, that uses predefined mapping rules to mediate between two ontologies.

Communication Manager So far only simple message exchange predefined patterns are the only interaction styles supported.

Choreography Choreography does not exist in the initial version of WSMX, only initial set of interfaces have been defined.

WSMT WSMT does not exist in the initial version of WSMX, however standalone tools supporting mapping process or WSMO editor exist.

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:

⁸Spring framework: <http://www.springframework.org/>



- WSMX Mission Statement** [8]. The Mission Statement deliverable clearly specify the mission of the WSMX working group - to create an execution environment for the dynamic discovery, selection, mediation, invocation and inter-operation of SWS.
- WSMX System Functionality Scope** [20]. This deliverable aims provides an overview of expected functionality the system is going to provide with each subsequent release.
- Web Service Execution Environment - Conceptual Model** [5]. The conceptual model defines all concepts used in WSMX. This conceptual model is based on WSMO; if needed it will extend it by introducing several concepts.
- WSMX Execution Semantics** [19]. The WSMX Execution Semantics deliverable defines the operational behavior 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.
- WSMX Architecture and System Design** [21]. This deliverable presents the architecture of WSMX, this being the basis for the implementation. The deliverable explains the architectural decisions that were made, the design of the system, and describes the various components.
- WSMO Discovery** [9]. The WSMO Discovery deliverable describes how discovery is dealt within WSMO. Different techniques for web service discovery are discussed including mediation support needed for different approaches to web service discovery. In detail, logic based discovery of web services is discussed.
- WSMX Data Mediation** [14]. The WSMX Data Mediation deliverable explains how mediation is dealt within WSMX. This deliverable describes several aspects related to WSMX 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.
- Processes Mediation in WSMX** [4]. This deliverable tackles the processes mediation problem, describing what process mediation means in the context of WSMX, and proposing a process mediation architecture.
- WSMX Grounding** [7]. This document describes how WSMX handles the translation to and from WSML to other data representations at the boundary of the environment.
- WSMX Documentation** [1]. WSMX Documentation provides the description of a step by step WSMX installation and examples of its usage.
- WSMX Use Cases** [6]. The Use Cases deliverable exemplifies several usage scenarios of WSMX. Three possible use cases are identified, namely a Business-to- Consumer (B2C), a Business-to-Business (B2B) and an Application-to-Application (A2A) scenario. Showcase how WSMX can be utilized for them is described.



Web Service Modeling Toolkit (WSMT) [10]. The purpose of this document is to outline the Web Services Modeling Toolkit (WSMT). WSMT is a framework for the rapid deployment of graphical administrative tools, which can be used with WSMO, WSML and WSMX.

WSML Editor [11]. This deliverable describes the WSML Editor plugin for the Web Service Modeling Toolkit. The WSML Editor is a graphical tool for editing Ontologies, Goals, Mediators and Web Services described in the WSML family of languages.

WSMX Monitor [12]. This document outlines the WSMX Monitor plugin for the Web Service Modeling Toolkit. The WSMX Monitor is a graphical tool for monitoring the state of each of the WSMX components within the WSMX system, along with the system itself.

5 Related Systems

Implementations based on SWS concepts are nowadays subject of extensible research. Apart from WSMX as the reference implementation of WSMO, other implementations based on SWS concepts also exist such as IRS[15], OWL-S[16] or METEOR-S[17]. It is the intention of WSMX to be interoperable with such systems where possible.

The main components of IRS (Internet Reasoning Server) are the IRS Server, the IRS Publisher and the IRS Client. The IRS server holds descriptions of SWS. IRS Publisher links Web services to their semantic descriptions within the IRS server and generates a wrapper which allows web service to be invoked. An IRS user asks through IRS Client for a task to be achieved, and accordingly the IRS server selects and invokes an appropriate Web service. Recently, IRS-III provides infrastructure for creating WSMO-based SWS, building upon the previous implementation of IRS.

A set of OWL-S tools exists rather than complete execution environment based on OWL-S concepts. For example, OWL-S Editors allow maintaining OWL-S service descriptions, OWL-S Matcher provides algorithm for different degrees of matching for individual elements of OWL-S, OWL-S Axis plugin provides a suggestion for service providers how to provide service description using OWL Services, etc.

The main components of METEOR-S are Abstract Process Designer, Semantic Publication and Discovery Engine, Constraint Analyzer, and Execution Environment. Abstract Process Designer allows a user to create the control flow of the process using BPEL constructs. Semantic Publication and Discovery Engine provides semantic matching based on subsumption and property matching. Constraint Analyzer dynamically selects services from candidate services, which are returned by the discovery engine. Execution environment performs actual late binding of services returned by the constraint analyzer and converts abstract BPEL to executable BPEL.

6 Conclusion

Concluding, we design and build an execution environment for the dynamic discovery, selection, mediation, invocation and interoperation of SWS. Through our research on WSMX we aim to provide a guideline and justification for an



architecture for the SWS systems. We present how SWS architecture should look like, we define its execution semantics, and any aspect we perceive compulsory in any SWS related systems. In the initial stage we focus on delivering a component-based architecture, ensuring a rigid foundation for later extension. We design all the components based on WSMO conceptual model. In a later stages these components will be extended to fully support WSMO and to provide a fully fledged enterprise system for execution of the SWS.

7 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] David Aiken and Maciej Zaremba. D22v0.1 wsmx documentation. Technical report, Dec 2004. <http://www.wsmo.org/2004/d22/v0.1/20041210/20041210.pdf>.
- [2] T. Bellwood, L. Clment, and C. von Riegen. *UDDI Specification Version 3.0.1*. UDDI Spec Technical Committee, October 2003. http://uddi.org/pubs/uddi_v3.htm.
- [3] R. Chinnici, M. Gudgin, J. Moreaum, and S. Weerawarana. *Web Services Description Language (WSDL) Version 1.2*. World Wide Web Consortium, March 2003. <http://www.w3.org/TR/wsd120/>.
- [4] Emilia Cimpian and Adrian Mocan. D13.7 v0.1 processes mediation in wsmx. Technical report, Jan 2005. <http://www.wsmo.org/2005/d13/d13.7/v0.1/>.
- [5] Emilia Cimpian, Adrian Mocan, Matthew Moran, Eyal Oren, and Michal Zaremba. D13.1v0.3 web service execution environment - conceptual model. Technical report, Dec 2004. <http://www.wsmo.org/2004/d13/d13.1/v0.3/>.
- [6] Armin Haller. D13.6v0.1 wsmx use cases. Technical report, Dec 2004. <http://www.wsmo.org/2004/d13/d13.6/v0.1/20041205/>.
- [7] Armin Haller. D26v0.1 wsmx grounding. Technical report, Dec 2004. <http://www.wsmo.org/2004/d26/v0.1/20041206/>.
- [8] Armin Haller and Michal Zaremba. D7.3v1.0 mission statement - wsmx. Technical report, Jan 2005. <http://www.wsmo.org/2005/d7/d7.3/v1.0/20050109/>.
- [9] Uwe Keller, Rubn Lara, Axel Polleres, Ioan Toma, Michel Kifer, and Dieter Fensel. D5.1v0.1 wsmo web service discovery. Technical report, Nov 2004. <http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112/>.



- [10] Mick Kerrigan. D9.1v0.1. web service modeling toolkit (wsmt). Technical report, Jan 2005. <http://www.wsmo.org/2005/d9/d9.1/v0.1/20050127/>.
- [11] Mick Kerrigan. D9.2v0.1 wsml editor. Technical report, Jan 2005. <http://www.wsmo.org/2005/d9/d9.2/v0.1/20050131/>.
- [12] Mick Kerrigan. D9.3v0.1 wsmx monitor. Technical report, Jan 2005. <http://www.wsmo.org/2005/d9/d9.2/v0.1/20050131/>.
- [13] N. Mitra. *SOAP Version 1.2 Part 0: Primer*. World Wide Web Consortium, June 2003. <http://www.w3.org/TR/soap12-part0/>.
- [14] Adrian Mocan and Emilia Cimpian. D13.3v0.2 wsmx data mediation. Technical report, Jan 2005. <http://www.wsmo.org/2005/d13/d13.3/v0.2/>.
- [15] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS: A Framework and Infrastructure for Semantic Web Services. In *2nd International Semantic Web Conference*, 2003.
- [16] OWL Services Coalition. OWL-S: Semantic Markup for Web Services. Available from <http://www.daml.org/services/owl-s/1.1/>, 2004.
- [17] Prof. Amit Sheth, Prof. John Miller, Kunal Verma. METEOR-S: Semantic Web Services and Processes. Available from <http://lsdis.cs.uga.edu/Projects/METEOR-S/>, 2004.
- [18] D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology Standard. Technical report, WSMO Working Draft, 2004. Available from <http://www.wsmo.org/2004/d2/v02/20040306/>.
- [19] Maciej Zaremba and Eyal Oren. D13.2v0.2 wsmx execution semantics. Technical report, Feb 2005. <http://www.wsmo.org/2005/d13/d13.2/v0.2/>.
- [20] Michal Zaremba. D23v1. wsmx system functionality scope. Technical report, Nov 2004. <http://www.wsmo.org/2004/d23/v1/>.
- [21] Michal Zaremba and Matthew Moran. D13.4v0.2 wsmx architecture. Technical report, Jan 2005. <http://www.wsmo.org/2005/d13/d13.4/v0.2/>.