WSMX Deliverable

D13.8 v0.2
Integration of WSMX with other SWS Systems

WSMX Working Draft – 12th October 2005

Authors:
Tomas Vitvar, Matthew Moran, Michal Zaremba, Adrian Mocan

Editors:
Tomas Vitvar

This version:
`http://www.wsmo.org/TR/d13/d13.8/v0.2/20051012`
Latest version:
`http://www.wsmo.org/TR/d13/d13.8/`
Previous version:
`http://www.wsmo.org/TR/d13/d13.8/v0.1/20050614`

Abstract

This deliverable describes the relationship between the Web Services Execution Environment (WSMX) and other implemented frameworks for working with Semantic Web Services. In particular, we look at how WSMX and these other systems can interoperate with each other. In the case of IRS III, both systems are compliant to the Web Services Modelling Ontology (WSMO). Development of a shared common API is already well underway.

Contents

1 Introduction

The Web Service Execution Environment (WSMX) provides a framework for the discovery, selection, mediation and invocation of Semantic Web services. In other words WSMX provides the middleware that allows requesters and providers of Web services to find and communicate with each other based on the semantic descriptions of their functional (*what they have to offer*) and non-functional (*requirements and constraints on their offerings*) properties. WSMX is based on the conceptual model provided by the Web Services Modeling Ontology (WSMO) [10] which describes various aspects related to Semantic Web services. WSMO descriptions are represented using the Web Services Modelling Language (WSML) [3].

There are other frameworks and systems that apply Semantic Web technology for the execution of Web services to a greater or lesser extent. This deliverable examines the possibilities for interoperability between WSMX and these systems. In some cases this is already possible to a significant degree while in other cases work is underway and the outlook for success quite positive due to a common underlying approach. We adopt a simple framework for the discussion. Each SWS system is described in terms of two main questions:

- **What is it?** We look at the functionality offered, how semantics are supported and the system architecture.
- **How can it be integrated with WSMX?** We can categorize integration in three ways: (i) both understand WSML (design-time integration), (ii) one system can invoke the other (run-time integration) and/or (iii) they share common component interfaces (component portability).

Section 2 provides a brief overview of WSMX for completeness. Section **??** looks at the Internet Reasoning Service (IRS) developed by the Knowledge Media Institute (KMI) at the Open University (OU). Section **??** looks at the METEOR-S project from the LSDIS Lab, Department of CS, University of Georgia, US. Section **??** discusses the OWL-S Virtual Machine developed at the Intelligent Software Agents and Web Technologies Lab, Carnegie Mellon University (CMU), US. The deliverable concludes with a summary.

2 Overview of WSMX

WSMX provides a reference implementation for WSMO in the form of a Service Oriented Architecture (SOA): a set of loosely coupled collaborating software components, each with well defined public interfaces that together perform a set of tasks. The semantics of the component interfaces describe the functionality offered by the components but not how this functionality must be implemented. Communication between components is based on the exchange of descriptive messages defined in terms of the conceptual model provided by WSMO. The current release of WSMX uses an events-based mechanism implemented using tuple spaces where the message exchange between components is based on a publish and subscribe mechanism. The implementation of the messaging mechanims is transparent to the implementation of the functional components. From their perspective message exchange appears as simple java invocations to the public interfaces of other components. This is described in detail in the WSMX architecure document at [13].

WSMX does not assume that service providers and requesters will user the same ontology to describe their data or the parts of their business processes that they wish to make public. It would be convenient if all Web users subscribed to a single global ontology but this is unlikely to happen. Instead WSMX adopts the principle described in the Web Service Modelling Framework (WSMF) [4], of providing a scalable mediation service between heterogeneous data and process types. Mediators act like bridges that map between ontologies used by different parties to an interaction.

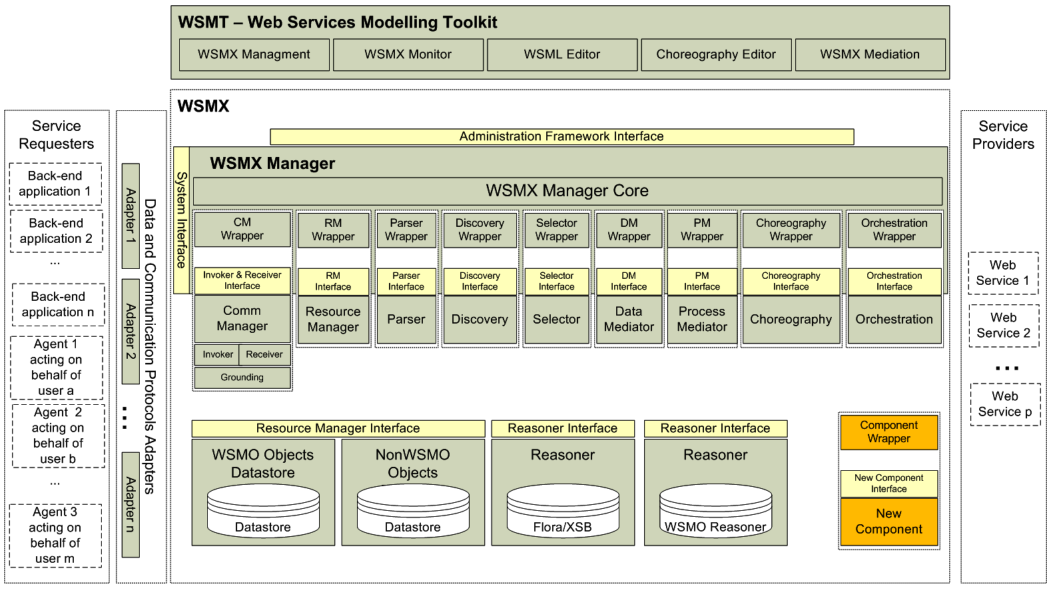

Figure 1: WSMX Architecture

A very brief description of the functionality of key WSMX components in figure 1 follows while a more detailed description is available at [13]. The **WSMX Manager Core** manages the events engine, internal workflow engine and the loading of registered components at start-time. The **Resource Manager** manages the persistent storage of both WSMO and non-WSMO entities. The **Parser** parses WSML documents into equivalent WSMO4J [1] objects. **Discovery** is responsible for finding Web services whose capability matches

[1]http://wsmo4j.sourceforge.net/

the goal provided by the service requester. Where multiple candidate Web services are identified, the **Selector** selects the Web service that provides the best match for the goal based on service requester preferences. During discovery or service execution, the **Data Mediator** and/or **Process Mediator** may be required to mediate between data and behaviour from heterogeneous sources. The **Communication Manager** is responsible for dealing with all aspects of sending and receiving messages to and from WSMX. The **Choreography** component manages the conversation between WSMX and Web services while the **Orchestration** component deals with the creation of new services based on the composition of existing ones. Reasoning support is provided using Flora-23. The **Web Service Modelling Toolkit** (WSMT) [5] is a framework for the deployment of graphical administrative tools, which can be used with WSMO, WSML and WSMX.

3 IRS III

3.1 Overview

3.1.1 Functionality

The Internet Reasoning Service (IRS) [8] is a framework for Semantic Web Services that supports the publication, location composition and execution of Web services based on their semantic descriptions. IRS supports the conceptual model defined by WSMO and also provides mappings for service descriptions provided in OWL-S. The next paragraphs provide a brief description of the main functionality offered by IRS.

Publication. Publication in IRS has two roles. The first is where a Web service represented by a URI endpoint is associated with a semantic service description known to IRS. The second is where standalone Java or Lisp code is wrapped to make it appear as a Web service and then, as in the first case, the service is associated with a semantic service description known to IRS. Once a service has been published to IRS it is available to be used in the achievement of a user goal.

Discovery. IRS has its foundation in an earlier project called IBROW [1] which made the distinction between *tasks* that need to be solved and *problem solving methods* that "provide abstract, implementation-independent descriptions of reasoning processes which can be applied to solve tasks in specific domains". Bridges were then defined to provide mappings to counter problems of heterogeneity. Adopting the WSMO conceptual model, tasks in IRS are modelled as *goals* while problem solving methods are modelled as *services*. Discovery in IRS is based on matching the pre- and post-conditions defined in the semantic descriptions of goals and services known to the IRS server.

Composition. Composition in IRS is goal based. Where a goal can not be achieved by a single service, it is decomposed into smaller goals each of which can either be mapped directly to a specific service description or can be resolved by the IRS server at runtime. The approach uses mediators to handle the problems of data heterogeneity.

Execution. The IRS server handles the invocation of Web services based on mapping the semantic description of the service choreography to the actual implementation of the service. This part of the IRS functionality is hidden from clients of IRS.

3.1.2 Semantic Support

IRS-III is compliant with the conceptual model defined by WSMO. It extends the WSMO defintion of Capability to include support for inputs and outputs. This can also be mapped directly to the WSMO conceptual model as inputs and output can also be represented as part of the pre- and post-condtions for a service or goal. Internally IRS-III uses OCML as the representation language for descriptions and based on which its reasoner operates. Work is underway with IRS-III to support OWL-S descriptions by providing a mapping to the the internal IRS representation of OCML.

3.1.3 Architecture

The main components of IRS are the IRS Server, the IRS Publisher and the IRS Client. The IRS Server stores the descriptions of goals, mediators and web services along with domain ontologies. Discovery, composition, mediation, reaosning and invocation are all controlled by the IRS Server. The IRS Publisher carries out the tasks required for publication as described above. Finally, the IRS Client provides a user-interface for goal-based Web service invocation.

3.2 Integration Points with WSMX

Both IRS and WSMX are implemented frameworks for Semnatic Web services based on the conceptual model defined by WSMO. This provides a strong basis for common ground between the systems. In terms of IRS, this meant defining system interfaces that could accept messages represented using the Web Services Modelling Language (WSML) [3]. This provided *design-time integration* in the sense that IRS could now exchange WSML messages with WSMX. Both systems used the same open-source software to parse the messages and represnt them internally as Java objects (WSMO4j [2]).

To support system interoperability between WSMX and IRS, the current releases of both systems each support a common public interface to their functionality. This is *run-time integration* following the pattern exemplified by the odbc and jdbc specifications with respect to relational database technology. It means that a client of a Semantic Web services execution environment can be transparently configured to use either IRS or WSMX without needing to change the client code. This holds equally true for instances of WSMX and IRS themselves as clients of instances of each other (or another compliant system). The common API is defined as an extension to the WSMO4j library.

The following subsections provide a short overview of the functionality and signatures of the common API. We look at the interfaces providing access to the functionality of discovery, goal-based service invocation, direct service invocation and registry-invocation. For convenience, the interface method descriptions are given in the syntax of Java Standard Edition 5.0 (J2SE 5.0) [3].

Many of the methods return an instance of the Context class as defined in the API. The notion of Context is used to represent a unique handle for identifying the context of communication between the Semantic Web Services execution enviornment and another entity. For example, a client wishing to use the environment to perform a goal-based service invocation, would initiate a conversation with the environment by sending a message containing the description of the goal that they wish to acheive. On receipt of this message the environment would create a unique Context for that specific converstaion and return it to the client. All subsequent interactions as part of the conversation must then include this Context. In this sense the Context is used as a shared token between the client and the environment to identify the state of the conversation on both sides.

3.2.1 Discovery

Discovery allows the client to find Web services that match a specific goal description. The method accepts as input a WSMLDocument object which should contain the WSML description of the goal. The method signature is:

```
public Context getWebService (WSMLDocument wsmlMessage)
```

Note: The Context object returned is intended to be used by the client of the environent as a handle to the conversaion started with the environment. Any subsequent messages received by the environment with this Context would

[2]http://wsmo4j.sourceforge.net/
[3]http://java.sun.com/j2se/1.5.0/

allow it to refer to the (possibly zero) WebService descriptions matched during the discovery operation. There are still open questions on how the client can get direct access to discovered service descriptions.

3.2.2 Goal-based Service Invocation

The environment will carry out all activities required for service invocation (discovery, mediation etc.) based on the input of user goal defined in WSMO.

```
public Context achieveWebService (WSMLDocument wsmlMessage)
```

3.2.3 Direct Service Invocation

The client of the environment can invoke a WebService directly by sending a WSML document identifying the Web service and the data that is required for input to the Web service.

```
public Context invokeWebService (WSMLDocument wsmlMessage)
```

Additionally the client can send a message to a Web service where a conversation has already been initiated (Context has been returned to client). in this case the client provides a WSML document containing the message to be sent to the Web service as well as the Context that identifies which conversation this action should be part of.

```
public Context invokeWebService (WSMLDocument wsmlMessage, Context context)
```

3.3 Next Steps

The common interface is at an early stage but provides the first steps towards run-time interoperability between WSMX and IRS. The next steps are to test and refine the interface ensuring that it covers the required common observable behaviour of the systems. Additional further steps are to have common interfaces defined for the functional components making up both IRS and WSMX. This would bring the advantage of componet-level interoperability where an IRS component could be seamlessly plugged into WSMX and vice versa.

4 METEOR-S

METEOR-S project of the LSDIS Lab [4] proposes the application of semantics to existing Web service technologies. In particular the project endeavours to define and support the complete lifecycle of Semantic Web service processes. Their work includes extending WSDL to support the development of Semantic Web services using semantic annotation from additional type systems such as WSMO and OWL ontologies. A similar approach to data mediation is taken in both WSMX and METEOR-S, based on using a common data representation format. METEOR-S proposes an enhancement of UDDI to facilitate semantic discovery as well as a framework for SWS composition. In contrast to WSMX, METEOR-S is not based on an overall SWS conceptual model. Mediators and goals are not modelled as first class citizens. Architecturally, METEOR-S is a collection of related discrete tools rather than the single encapsulated architecture provided by WSMX.

The current implementation of METEOR-S allows for the (i) the creation of WSDL-S descriptions from annotated source code, (ii) the automatic publishing of WSDL-S descriptions in enhanced UDDI registries, and (iii) the generation of OWL-S descriptions, from WSDL-S, for grounding, profile and service. We take a look at each of these aspects in the next sections. Some of the text for these descriptions is taken from the upcoming workshop paper at [6]

4.1 Functionality

4.1.1 WSDL-S: Development of Semantic Web Services

WSDL-S [12] is a joint proposal between LSDIS Lab and IBM and represents a lightweight approach to adding semantics to Web services by providing a simple extension to the meta-model of draft release for the WSDL 2.0 specification [2]. The extension allows takes advantage of the WSDL support for multiple typesystems (e.g. XSD, WSMO and OWL-S) and provides a mechanims to specify constraints on the operations that make up the units of functionailty offered by WSDL service interfaces.

4.1.2 Publication and Discovery

The publication and discovery (MWSDI) [7] module provides support for semantic publication and discovery of Web services. It provides support for discovery in a federation of registries as well as a semantic publication and discovery layer over UDDI.

4.1.3 Composition of Web Processes

The composition module [11] consists of two main sub-modules the constraint analysis and optimization sub-module and the execution environment. The constraint analysis and optimization sub-modules deal with correctness and optimization of the process on the basis of quality service constraints. The execution environment provides proxy based dynamic binding support to BPWS4J execution engine for BPEL4WS.

[4]http://lsdis.cs.uga.edu/projects/past/METEOR/

4.1.4 Use of Semantics

METEOR-S advocates the use of a minimal bottom-up approach when annotating Web services with additional semantics. WSDL-S is agnostic to the type system used for adding semantics. Up to now it has been limited to the use of OWL which have been sufficient for representing the data semantics of inputs and outputs but limited when representing the pre-conditions, post-conditions, assumptions and effects of a Web service.

4.2 Integration Points with WSMX

4.2.1 Grounding to WSDL

WSMO and METEOR-S propose solutions towards Semantic Web services, but most existing Web services have to be invoked using SOAP and their interfaces are described using WSDL. Therefore, grounding is about filling the gap between the WSMO semantic descriptions of Web services and the classical technologies used today. Basically, two main tasks can be distinguished: first to determine which WSDL operation corresponds to the piece of functionality to be consumed (semantically described in the WSMO Web services capability); second to prepare the data to be used in the invocation as required by the types definition of inputs in the invoked operation. WSMX was confronted with this problem even from the first release when a hard-coded solution was adopted to map semantic descriptions of the Web services with specific WSDL operations. As WSMX became more mature, dynamic and coherent solution were required to enable the full integration of classical Web services in this framework for semantic Web services.

Grounding of the WSMO service description is required from two perspectives: data and observable behaviour. The solution envisioned implies the creation of a set of transformations between a given source XML Schema (or even directly between XML data) and a target WSMO ontology. A ”lifting” operation is applied to the XML Schema to create a corresponding WSMO ontology. This approach acts on a semantic level, but its effectiveness depends on the techniques used in the lifting and ontology mapping processes which might represent two sensitive points to errors and loss of information. As a consequence, the solution proposed by LSDIS-IBM group as part of the METEOR-S project could represent a possible intermediate alternative.

WSDL-S is a lightweight approach for adding semantics to Web services [WSDL-S]. It allows semantic representation of inputs, outputs, preconditions and effects of Web service operations, by adding extension to WSDL, which is a well accepted standard in the industry. In principle, WSDL-S allows semantic annotations using domain models, which are agnostic to the domain representation language. It means that WSMO ontologies can be used in the annotation process and directly link the WSMO semantic description of Semantic Web services with WSDL descriptions. In this way, the annotations of the inputs and outputs in WSDL will represent concepts in a WSMO ontology and the preconditions and effects associated with WSDL operations will point to preconditions and effects from the definition of a WSMO Web service. As Web service specifications in WSMO refer to assumptions and post-conditions as well, additional extensions to the WSDL operations are required to accommodate these features too - the new extensions would be similar with the existing ones for preconditions and effects. By this, one Web service operation will correspond to a Semantic Web service as described in WSMO; if it is desired that more than one operations to be associated to a Semantic Web service than additional annotations are required to link a specific operation with elements described in the choreography of the Web service.

4.2.2 Publication and Discovery

The main purpose of the service discovery in WSMX infrastructure is to provide functionality on matching of usable SWS from service providers with the goals of service requesters. When matching SWS, a number of possible services could be discovered whose capabilities satisfy criteria specified in a goal of a requester. WSMX provides Selection and Negotiation components allowing refinement the list of services based on their non-functional and functional properties respectively. Currently, WSMX publishes services to a Resource Manager together with all the WSMO and non WSMO related entities (non WSMO related entities are required by the run environment). The peer-to-peer infrastructure of semantically enhanced registries provided by METEOR-S could offer a useful solution to address WSMX requirements for the distributed persistent storage.

The current infrastructure of WSMX defines a Resource Manager to be responsible for management of repositories to store definitions of any WSMO (Semantic Web services, goals, ontologies and mediators) and non-WSMO related objects (e.g. mappings, messages, WSDL definitions, etc.). Depending on the scope of information stored in the underlying registries, WSMX distinguishes registries to be local or 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 to particular components, not to everybody) whereas, the global registry could be shared across several domains (e.g. registries of SWS or Goals descriptions). While both stores data, the accessibility to this data might considerably differ between registries. The Resource Manager remains the only entry point for them (it is not possible to access any of the registries directly, but only through the Resource Manager). The Resource Manager acts as a placeholder for the future peer-to-peer infrastructure of global registries although the implementation so far is based on a single relational database. To build the peer-to-peer infrastructure for global registries, we might either assume replication of WSMO entities across all registries and data stores or some sophisticated storage mechanism allowing us to find WSMO entities without replicating them.

The METEOR-S Web services Discovery Infrastructure (MWSDI) has been already proposed by METEOR-S and with some modification could suit WSMX requirements. METEOR-S proposes a specialized ontology called a Registries Ontology at the registries level. This ontology aims to map each registry to a specific domain. By simply providing an obligatory property describing the type of the registry we should easily classify WSMX registries as Web services, Ontologies, Goal and Mediators registries. Additionally to it, we could use the METEOR-S mechanism to provide a richer semantic description to distinguish the types of entities stored in the registry, e.g., we could have Ontology registries with EDI related ontologies, or Goal registries with goals from travel industry.

5 OWL-S Virtual Machine

The OWL-S Virtual Machine (OWL-S VM) provides a general purpose Web service client for the invocation of a Web service based on the process model provided by OWL-S descriptions. The architecture consists of components for executing the OWL-S process model, the grounding, and for making the Web service invocation. The OWL-S VM provides a first implementation, and therefore proof-of-value, of the OWL-S process model but only addresses a subset of the functionality offered by WSMX. For example, WSMX focuses on providing an environment where multiple semantically described services can interact. Data and process mediators are first class citizens, based on the assumption that independent services will exhibit, not only data, but also behavioural independence.

5.1 Functionality

There are two principle functional components in the OWL-S VM architecture - (i) the OWL-S VM Processor and (ii) the Web Service Invocation Module. These are briefly described in the next sections.

5.1.1 OWL-S VM Processor

This derives and executes the sequence of processes described in the Process Model of an OWL-S service description. The execution of the process descriptions follows a formal semantics defined in [9]

5.1.2 Web Service Invocation Module

This implements a set of rules as XSLT to ground the OWL-S description of the Web service to the corresponding WSDL service description. Once the grounding has taken place the OWL-S VM uses the Axis SOAP [5] implementation to actually invoke the service.

5.2 Integration Points with WSMX

WSMX and the OWL-S VM are based on substantially different conceptual models and derived formal langauges. Based on a common subset of WSML and OWL, it is possible that common domain ontologies could be made available for both systems. The scope and functional characteristics of both environments are considerably different. It is not clear how a client would program would interact with OWL-S VM to achieve a particular task as the OWL-S VM itself acts as primarily as a client of Semantic Web services defined in terms of OWL-S.

Up to the present, there has been little opportunity to investigate the possibilities of how WSMX and OWL-S might interoperate due partly to the fact that the source code for the OWL-S VM is not publicly available.

[5]http://ws.apache.org/axis/

6 Summary and Outlook

In this deliverable, we have taken an initial look at the possibilities for interoperation between WSMX and the other most well known execution environments for Semantic Web services. The IRS-III platform shares a common conceptual model and system interface with WSMX, facilliating a high degree of interoperability. The METEOR-S research offers potential specific co-operation opportunities in the area of grounding to WSDL, peer-to-peer based repositories and service composition. The OWL-S VM provides a specific client capable of executing the process models of Web services defined with OWL-S. At present, there is little scope for interoperation between WSMX and the OWL-S VM.

References

[1] V.R. Benjamins, E. Plaza, E. Motta, D. Fensel, R. Studer, B. Wielinga, G. Schreiber, Z. Zdrahal, and S. Decker. An intelligent brokering service for knowledge component reuse on the world-wide-web. In Gaines and Musen, editors, *11th Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Canada, 1998.

[2] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreaum, Sanjiva Weerawarana, and Jeffrey eds. Schlimmer. Web Services Description Language (WSDL) Version 2.0. World Wide Web Consortium, March 2003. Available from `http://www.w3.org/TR/2004/WD-wsdl20-20040326/`,.

[3] Jos de Bruijn. D16 The WSML Specification. Technical report, WSML Working Draft v0.2, 2005. Available from `http://www.wsmo.org/TR/d16/v0.2/`.

[4] D. Fensel and C. Bussler. The Web Services Modeling Framework (WSMF). *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[5] Mick Kerrigan. D9.1v0.2 The Web Services Modelling Toolkit (WSMT). Technical report, 2005. `http://www.wsmo.org/TR/d9/d9.1/v0.2/`.

[6] Kunal Verma and Adrian Mocan and Michal Zaremba and Amit Sheth and John A. Miller. Linking Semantic Web Services Efforts. In *Second International Workshop on Semantic and Dynamic Web Processes (SDWP), 2005*, 2005.

[7] J. Miller and S. Oundhakar. METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web services. *Journal of Information Technology and Management, Special Issue on Universal Global Integration*, 6(1):17–39.

[8] 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.

[9] M. Paolucci, Anupriya Ankolekar, Naveen Srinivasan, and Katia Sycara. The daml-s virtual machine. In *The Semantic Web - ISWC 2003, LNCS 2870, Springer-Verlag*, pages 290–305, 2003.

[10] D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology (WSMO). Technical report, 2004. Available from `http://www.wsmo.org/TR/d2/v0.2/`.

[11] K. Sivashanmugam, J. Miller, K. Verma, and A. Sheth. Framework for Semantic Web Process Composition. volume 9, pages 71–106. M.E. Sharpe Inc., 2005.

[12] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Service Standards. *First International Conference on Web Services*, 2003.

[13] Michal Zaremba, Matthew Moran, and Thomas Haselwanter. WSMX Architecture v0.2. Technical report, 2005. Available from `http://www.wsmo.org/TR/d13/d13.4/v0.2/`,.

7 Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; 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 RW^2 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.

8 Appendix B. Changelog

2005.06.14 - Matthew Moran

- Updated the introduction and overview of WSMX sections
- Added sections for IRS, METEOR-S and OWL-S VM

2005.04 - Matthew Moran

- Document created