Using WSMX to bind Requester & Provider at Runtime when Executing Semantic Web Services

Matthew Moran, Michal Zaremba, Adrian Mocan, Christoph Bussler

Digital Enterprise Research Institute, DERI, Ireland
{matthew.moran, michal.zaremba, adrian.mocan, chris.bussler}@deri.org

Abstract. Since its introduction, Web Service technology has promised to revolutionize how businesses can release their services and technology to the world over the Internet. However, the widely accepted standards underpinning Web Services do not solve the problem of how autonomous, heterogeneous services can be discovered, mediated and invoked at runtime by service requesters. Requesters can locate services in public registries but, at design time, they still have to interpret the provider's intent of what the service means and how it should be invoked. The Web Services Modelling Execution Environment, WSMX, is a framework that facilitates service requesters in discovering, selecting, mediating and invoking Web Services offered by providers at runtime.

1 Introduction

Despite being based on widely accepted standards, Web Service technology has not realized its potential for revolutionizing internet-based integration between companies. One of the significant stumbling blocks for existing Web Services framework is that there is no common understanding of the meaning of the data input and output by Web Services. The implication is that automatic discovery and invocation of Web Services is not possible in an open environment. As a consequence service requesters need to know and bind to service providers at design time.

WSMX is an implemented software framework that allows web services whose semantics have been described in a formal language to be discovered, selected, combined and invoked. Semantics is the formal meaning of various aspects of Web Services that allow machines to reason about, and invoke Web Services with a minimum or no human intervention. WSMX is a reference implementation for the Web Service Modelling Ontology, WSMO [1], and has its foundation in the Web Service Modelling Framework, WSMF [2]. In particular, WSMX adopts the principles advocated in the WSMF of strong decoupling between system components and a strong scalable mediation service that enables everybody to speak to everybody else.

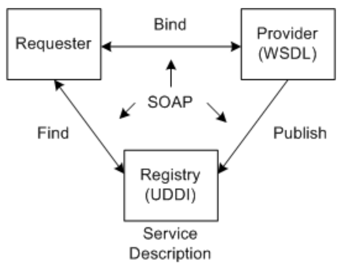

Figure 1. IBM WS Model [3]

The Web Service architecture proposed by IBM in [3] is illustrated in figure 1 and is typical of current Web Service technology. A service provider syntactically describes his service using WSDL[1] and publishes the service description to a service registry. Service requesters can search the registry for a particular service and once located, can write code in whatever language they wish, to bind to this service. In this model web services are based on the open standards of HTTP[2] for communication, SOAP[3] for messaging, WSDL for service description, and UDDI[4] for discovery. This model of Web Services has the important feature of being language independent. A service developed in C++ and published to UDDI by a provider may be accessed by a Java application written by a service requester. However, even with the standards mentioned above, the process required by a service requester to locate one or more Web Services and then to invoke these services remains a manual task requiring significant up-front knowledge or assumptions about the services.

The nature of the manual steps a service requester has to take can be best illustrated through a simple example of locating and using a Web Service to book a train ticket for travel in Germany or Austria. The first step is to search for a suitable service in a UDDI service directory. The requester might try looking for services with the name ‘Train Ticket Reservation’. If no services were located, they might try a search on ‘Book Train Tickets’ or even ‘Fahrschein Reservierung’. If a service is located in the registry, the service description can be checked to see if the service fits the requester’s requirements. However, as service descriptions provided in UDDI are informal, the requester needs to be able to interpret the meaning and assume that their understanding is the same as that intended by the service provider. Once the requester is satisfied with a web service, the associated WSDL document provides a syntactic description of the groundings available for the service and what input and outputs the service expects. The input and output messages are described in XML, optionally using an XML schema. Again, the requester must interpret the meaning of the parts of the input and output messages. To make an invocation of the Web Service, they may have to adjust their data to fit the service description. Software such as the SOAP implementation provided by Apache Axis[5] can generate client and server stubs based

[1] Web Services Description Language (WSDL) 1.2, W3C Working Draft, 3 March 2003
[2] Hypertext Transfer Protocol – HTTP/1.1, June 1999
[3] SOAP version 1.2, W3C Recommendation, 24 June 2003
[4] OASIS UDDI v2 specifications (http://www.oasis-open.org/)
[5] Apache Web Services Project. http://ws.apache.org/axis/

on the WSDL for a Web Service. The stubs in this context are Java[6] classes that can be used by the requester to invoke the web service. They shield the requester from the complexity of interpreting the WSDL by providing a Java API. However they do not change the fact that the service requester still has to interpret the meaning of the data required by the Web Service.

In this example the binding between the service requester and service provider is hard-wired at design time. If the requester wants to book another train ticket later for cities not supported by the initial Web Service, they will have to start the whole process of finding and binding to a suitable service again.

2 Requirements for Runtime Binding

Many of the problems identified in the previous section arise from the absence of formal semantic descriptions for all aspects of Web Services. This is particularly clear in the area of service discovery using UDDI and making the actual service invocation. Formal descriptions based on mathematical logic can be precisely interpreted and reasoned about by computer systems. In terms of Semantic Web technology, ontologies provide the model and vocabulary for creating such descriptions. An extensive discussion on ontologies and their importance to the Semantic Web is provided in [4]. Discovery systems based on formal service descriptions would allow the requester to provide a description of the service they required and would provide a matching service. However there is no guarantee that the data types of the input and output required by the matching service would match the data offered by the requester. A data mediation service that would allow runtime transformation from data described in the requester's ontology and data described in the provider's ontology is needed.

Formal descriptions need to be interpreted and used. An environment that can interpret the descriptions and carry out activities such as service discovery and invocation is required. The environment should provide simple interfaces that shield the environment users from the complexity of how the environment is implemented. The environment should have a modular construction where components have well defined functionality and interfaces. Ideally it should be possible to interchange implementations for specific components without having to rebuild the environment. The environment should have access to a repository for storing formal descriptions of services. The repository should allow Web Service descriptions to be added, modified or removed at any time without affecting the environment stability.

The overall goal of the environment should be to allow entities, acting as service requesters, to provide a formal description of a service they want to the environment. Based on this description the environment should carry out all the steps required to discover, and invoke a matching service.

[6] http://java.sun.com

3 Semantic Web Service Execution with WSMX

3.1 WSMX Overview

The Web Services Modelling Execution Environment, WSMX, [5] is a software framework that allows runtime binding of service requesters and providers. WSMX uses Semantic Web technology to discover, select, mediate and invoke Web Services based on the formal descriptions of various aspects of the Web Services themselves as well as the formal descriptions of service requester goals. To achieve their goal a service requester formally describes the goal and sends it to WSMX. Taking this goal description, WSMX searches its repository of Web Service descriptions for any Services whose descriptions match the goal. If more than one matching Web Service is found, WSMX selects the most suitable one based on preferences provided by the service requester. WSMX then ensures that the data provided in the service invocation is in the format that the Web Service expects. This operation of data mediation is possible as all the data representations used in both the goal and Web Service descriptions belong to concepts described in ontologies known to the WSMX environment.

The operations of discovering and selecting a Web Service, matching the requester goal and, then mediating data before finally invoking the Web Service, are all taken care of by WSMX. The service requester does not need to know in advance the data format expected by the service provider. As long as the data provided in the goal are defined by concepts in an ontology known to WSMX and, mappings have been defined between the concepts in that ontology and the ontology of the service provider, WSMX will be able to fulfil the goal. Additionally, if a new Web Service description that can better satisfy the requester goal is made available to WSMX at a later stage, WSMX will be able to select this Web Service to carry out the task without the service requester having to take any additional actions.

3.2 A Conceptual Model for WSMX

The operation of WSMX is made possible by its use of Semantic Web technologies, in particular the application of the Web Service Modelling Ontology, WSMO. WSMO is an ontology that describes various aspects related to Web Services. The top level concepts in WSMO are goal, web service, mediator and ontology. A full description of the conceptual model of WSMO is provided in [1]; a brief description follows in this paper for completeness.

Goals describe what a service requester wishes to achieve. The main elements of a goal are the post-condition and effect, both of which are described as logical expressions. Post-conditions describe what happens in terms of data within a system if the goal is achieved while effects are real world actions that follow from successful achievement of the goal. For example, in the case of the goal to buy a train ticket, the post-condition might be that the credit card billing system is invoked to debit the provided credit card number. The effect might be that a ticket is printed and posted to the service requester.

Web Services are defined in terms of the capability that they offer, the interfaces they provide, and the mediators that they can use. The capability of a web service is what is used when matching the requester goal. Capabilities have pre- and post-conditions as well as assumptions and effects. As with a goal, the pre- and post-conditions are functions on the input and output data of the service, while the assumptions and effects are based in the real-world. The discovery of a Web Service capability that matches a requester Goal happens at runtime. In the example of the train ticket booking, WSMX will check for the best service description for booking train tickets matching the requester's goal each time the goal is sent to WSMX. This means that if a new service provider provides a new services description for booking train tickets to WSMX, it becomes immediately available for use without any changes to the environment.

Mediators provide a way to transform data described in one conceptual model to data in a another conceptual model. Data mediation in WSMX is described later in this paper.

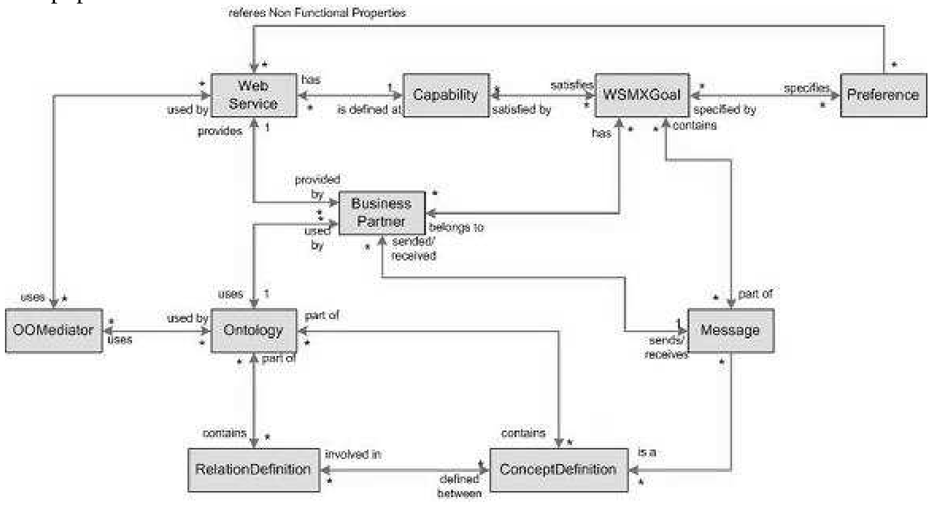

Figure 2. WSMX-O [7]

WSMX has a specific aim to address the problems associated with B2B integration. When identifying the concepts required for WSMX, a first step was to start with the conceptual model provided by WSMO and then extend it by looking at a simple problem from the B2B [6] domain. Two business partners want to communicate with each other in a business transaction e.g. one wants to send a purchase order to the other. Each business partner uses a different ontology for describing business documents and each of these ontologies may use one or more data mediators. This simple example led to the definition of a conceptual model for WSMX called WSMX-O [7], shown in figure 2, that contains some concepts not present in WSMO. These concepts are WSMXGoal which extends the Goal concepts of WSMO, Business Partner, Preference that a Business Partner may have, and Message. WSMX-O is described fully in [7].

There are several languages that can be used to represent the concepts of WSMO, as described in [8]. WSMX extends one of these languages WSML-Core [9] to include the additional concepts in WSMX-O.

3.3 WSMX Execution Semantics

The execution semantics of a system is the formal definition of the operation of that system. Execution semantics provide an unambiguous definition of the operational behaviour of the system. Once a system's operations have been modelled in this way, using a formal methodology, simulations can be run to check for deadlocks, livelocks and the effects of specific stimuli to the system. Deadlock means a situation in which two or more processes are prevented from continuing while each waits for resources to be freed by the continuation of the other. Livelock occurs where a system never reaches its termination state e.g. an endless loop. The execution semantics of the current version of WSMX are modelled using classical Petri nets and are described fully in [12].

Formal execution semantics allow the separation of the operational description of WSMX from the implementation of the individual components. In future versions of WSMX, this will provide the powerful feature of allowing the execution semantics to be modified and extended without requiring any software to be rewritten.

3.4 WSMX Architecture

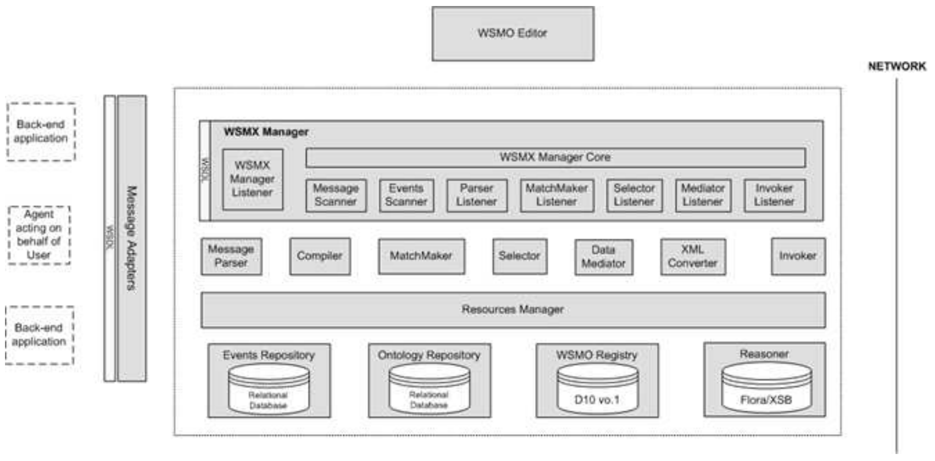

Figure 3. WSMX Architecture

A full description of the architecture of WSMX is described in [10] while a brief outline is included here for completeness. There are two operational aspects to the WSMX Architecture – compilation and execution. Compilation is the mechanism for making elements relating to Semantic Web Services ready for use by WSMX. WSMX-O descriptions of Web Services, Ontologies and Mediators, written in WSML, are compiled to WSMX. Execution, in the simplest case, means discovering and invoking the right web service to carry out a client goal. In more complex cases, execution could mean multiple discovery and invocation operations on different web services carried out in a controlled process. It is the compilation aspect that allows new service descriptions to be added to WSMX at any time independent of the execution aspect of the environment.

This paragraph briefly describes how each component plays its part in getting from a description of a requester's goal to the invocation of a matching Web Service. The architecture, shown in figure 3, has been implemented for the first version of WSMX. The **WSMO Editor** is used to create WSMO descriptions of web services, ontologies, mediators and goals. WSMX provides a WSDL interface to accept these descriptions and direct them to the **Compiler** component. Compilation means validating the descriptions and storing them persistently in the Ontology Repository. The **WSMX Manager** controls the operational flow of the system implementing the execution semantics. The **WSMX Manager** regularly scans for new messages. Once a new message representing a requester goal is picked up, it is translated into an internal persistent WSML representation by the Message Parser. The **Matchmaker** then carries out discovery to match the client goal to a capability of Web Services known to WSMX. The **Selector** component selects the web service that provides the best match for the goal based on service requester preferences. If necessary the **Data Mediator** mediates between the data (instance of ontology concepts) provided by the requester and those expected by the service provider. Once the data has been mediated, the **Invoker** makes the actual web service invocation on the selected web service using the mediated data.

3.5 Mediation in WSMX

As mentioned earlier, mediation is a key feature required to allow requester goals be matched to Web Services dynamically. WSMX does not assume that all web services share one conceptualization of the world. Rather it takes the view that the applications represented by Web Service interfaces are normally heterogeneous and autonomous. This gives rise to the need for mediation at data, process and business protocol levels. This version of WSMX starts with the problem of data mediation. An overview of data mediation in WSMX is hsown in figure 4 while a complete description can be found at [11].

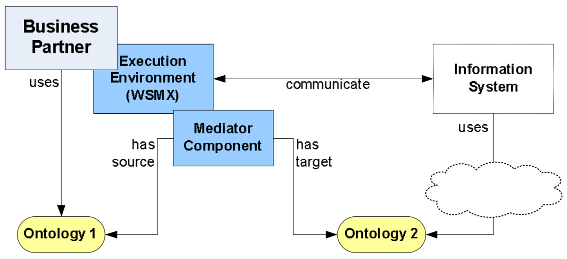

Figure 4. Data Mediation in WSMX [11]

The Execution Environment communicates with a Business Partner and an Information System each using a different ontology. When a set of instances have to be passed from the Business Partner to the Information System the Mediator Component is called to perform the transformation from source to the target format.

3.6 WSMX Implementation

The first version of WSMX architecture has been implemented through the open source project[7] and is described in detail at [13]. This implementation forms the backbone for an environment which can host components developed by anybody interested contributing to WSMX. The implementation of WSMX server remains minimal but complete. All components are in place with well defined interfaces and the executions semantics described for the system is operational. In the case of some components, only a simple implementation of functionality has been carried out to make these components operational.

The importance of implementing WSMX has been to prove the validity of the design, execution semantics and architecture. With this implementation of WSMX server, we also provide an implementation of a back-end application system that can take the role of a service requester by generating goal descriptions and sending these to WSMX via a Web Service interface. As a result, it is already possible to issue requester goals from back-end application, carry out discovery and data mediation before finally invoking a specific Web Service offered by a provider.

4 Related Work

The work referenced here is viewed from Semantic Web Services are still a relatively new research area but one in which there is considerable interest. In this section we compare other approaches to execution environments for Web Services particularly from the perspectives of when the binding of requester to provider is made

Biztalk Server[8] is Microsoft's platform for EAI and Business Process Management and is based on XML and Web Service technologies. Biztalk offers two core

[7] WSMX at sourceforge – http://sourceforge.net/projects/wsmx
[8] Microsoft Biztalk Server, http://www.microsoft.com/biztalk/

functions: a process execution engine and a messaging hub. Services used when executing processes with Biztalk must be identified in advance and bound to the process steps at design time.

IRS [14] is a Web Service execution environment developed by the Knowledge Media Institute of the Open University. The conceptual model for IRS and WSMX is very similar and both initiatives have roots in the IBROW project [18]. The most recent version, IRS3, is WSMO compatible and will be able to interoperate with WSMX. With IRS3, a service provider can create a WSMO service description that can be published against their service on the IRS3 server. The actual implementation of the service might be in Java or in another language such as Lisp. Once a Web Service description is available, a goal can be described in WSMO and bound to the published Web Service description using a mediator. The binding in this case is still at design time but the use of mediators to link goals and services removes the manual hard-wiring required for standard Web Services.

[15] implements an approach to dynamic binding of Web Services by augmenting the BPWS4J[9] implementation of BPEL4WS[10] with a semantic discovery service. This is prompted by the limitation of BPWS4J of not allowing dynamically discovered services to be assigned to partner roles. Additionally the authors point out that the BPEL4WS specification itself restricts the description of service partners to syntactic WSDL portType definitions. Their solution is to build a Web Service that can discover custom service partners at runtime based on DAML-S [16] descriptions of these services. This discovery service is then bound to a BPWS4J process at design time. This is an imaginative solution. In WSMX we build this functionality into the environment from the beginning.

Meteor-S WSDI [17] proposes a method of adding semantics to Web Services descriptions and then publishing those descriptions in UDDI registries. They provide tools to map WSDL descriptions to ontology-based semantic descriptions and propose that such descriptions could be used successfully for Web Service discovery if the user service requirement could be described in terms of concepts from the same ontologies. The main body of the paper describes an architecture and implementation for a P2P-based scalable infrastructure for accessing multiple registries. The design of public registries for hosting WSMO descriptions has not been considered yet in the WSMX design.

5 Conclusion

WSMX already provides a conceptual model, architecture and implementation of a Semantic Web Services execution environment that allows a goal provided by a service requester to be matched to a Web Service at runtime and for this service to be executed. WSMX combines the technologies of the Semantic Web and Web Services and is a reference implementation for WSMO. The first version of WSMX provides a robust service oriented architecture that provides a framework for ongoing research in the area of Semantic Web Services and in particular for WSMO.

[9] BPWS4J, http://www.alphaworks.ibm.com/tech/bpws4j

[10] BPEL4WS, http://www-106.ibm.com/developerworks/library/ws-bpel/

6 Acknowledgments

The work is funded by the Science Foundation Ireland under the DERI Lion project.

7 REFERENCES

1. Roman, D., Lausen, H., Keller, U.: The Web Service Modeling Ontology Standard (WSMO-Standard) v0.2, 6 March 2004, Digital Enterprise Research Institute. Available at http://www.wsmo.org/2004/d2/v0.2/20040306/
2. Fensel, D., Bussler, C.: The Web Service Modeling Framework, WSMF. Electronic Commerce Research and Applications, Vol. 1, Issue 2, Elsevier Science B.V.
3. IBM Web Services Conceptual Architecture, IBM, available at http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
4. de Bruijn, J.: Using Ontologies, Enabling Knowledge Sharing and Reuse on the Semantic Web, DERI Technical Report, DERI-2003-10-29, available at: http://www.deri.ie/publications/techpapers/documents/DERI-TR-2003-10-29.pdf
5. Oren E., Zaremba, M., Moran, M.: Overview and Scope of WSMX. WSMO Working Draft v0.1, 11 June 2004, Digital Enterprise Research Institute, available from http://www.wsmo.org/2004/d13/d13.0/v0.1/20040611/
6. Bussler, C.: B2B Integration, Concepts and Architecture, Springer-Verlag, 2003, ISBN 3-540-43487-9
7. Cimpian, E., Mocan, A., Moran, M., Oren, E., Zaremba, M.: WSMX Conceptual Model. WSMO Working Draft v0.1, Digital Enterprise Research Institute (DERI), available from http://www.wsmo.org/2004/d13/d13.1/v01
8. Oren, E., Lausen, H., de Bruijn, J.: Languages for WSMO. WSMO Working Draft, 3 August 2004, available at: http://www.wsmo.org/2004/d16/d16.0/v0.2/20040803/
9. de Bruijn, J., Foxvog, D., Oren, E., Fensel, D.: WSML-Core. WSMO Working Draft, 23 August 2004, available at: http://www.wsmo.org/2004/d16/d16.7/v0.1/20040823/
10. Zaremba, M., Moran, M., Oren, E., Cimpian, E., Mocan, A.: WSMX Architecture, WSMO Working Draft v0.1, 22 June 2004, available at: http://www.wsmo.org/2004/d13/d13.4/v0.1/20040622/
11. Mocan, A., Oren, E., Cimpian, E. Moran, M., Zaremba, M.: WSMX Mediation, WSMO Working Draft v0.1, 28 June 2004, Digital Enterprise Research Institute (DERI), available from http://www.wsmo.org/2004/d13/d13.3/v0.1/20040628/
12. Oren, E.: WSMX Execution Semantics. WSMO Working Draft v01, 31/5/04. available at: http://www.wsmo.org/2004/d13/d13.2/v0.1/20040531/index.pdf
13. Moran, M., Zaremba, M., Cimpian, E., Mocan, A., Oren, E.: WSMX Implementation, WSMO Working Draft v01, 19 July 2004, available at: http://www.wsmo.org/2004/d13/d13.5/v0.1/20040719/
14. Motta, E., Domingue, J., Cabral, L., Gaspari, M.: IRS-II A Framework and Infrastructure for Semantic Web Services in D. Fensel et al., (Eds.): The Semantic Web – ISWC 2003. Lecture Notes in Computer Science, Vol. 2870. pp 306-318, Springer-Verlag, Heidelberg (2003).
15. Mandell, D. J., McIlraith, S. A.: Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. D. Fensel et al., (Eds.): The Semantic Web – ISWC 2003. Lecture Notes in Computer Science, Vol. 2870. pp 227-240, Springer-Verlag, Heidelberg (2003).
16. DAML Services Coalition. DAML-S and OWL-S. http://www.daml.org/services/

17. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR–S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services, available at: http://lsdis.cs.uga.edu/lib/download/MWSDI.pdf
18. Benjamins, V. R., Plaza, E., Motta, E., Fensel, D., Studer, R., Wielinga, B., Schreiber, G., Zdrahal, Z., Decker, S.: An intelligent brokering service for knowledge component reuse on the World-Wide-Web. Gaines & Musen (Eds.), 11th Workshop on Knowledge Acquisition, Modeling and Management, Banff, Canada, 1998.