

# WSMX - A Semantic Service-Oriented Architecture

Armin Haller      Emilia Cimpian      Adrian Mocan      Eyal Oren  
Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway, Ireland  
firstname.lastname@deri.org

## Abstract

*Web Services offer an interoperability model that abstracts from the idiosyncrasies of specific implementations; they were introduced to address the increasing need for seamless interoperability between systems in the Business-to-Business domain. We analyse the requirements from this domain and show that to fully address interoperability demands we need to make use of semantic descriptions of Web Services.*

*We therefore introduce the Web Service Execution Environment (WSMX), a software system that enables the creation and execution of Semantic Web Services based on the Web Service Modelling Ontology. Providers can use it to register and offer their services and requesters can use it to dynamically discover and invoke relevant services. WSMX allows a requester to discover, mediate and invoke Web Services in order to carry out its tasks, based on services available on the Internet.*

## 1 Introduction

Interoperability problems are imminent in the domain of Business-to-Business (B2B) electronic commerce. When businesses engage in electronic trade they have to align their business applications with those of their trading partners; a Gartner report<sup>1</sup> shows that these integration costs account for around 40% of the average information technology budgets. Web Services were introduced as an answer to the continuously increasing need for seamless interoperability between systems in the B2B domain [9], offering a perfectly matching interoperability model abstracting from the idiosyncrasies of specific implementations.

However, several requirements from this domain are not fulfilled by current Web Service standards like SOAP<sup>2</sup>

and WSDL<sup>3</sup>. To fully exploit Web Services in the B2B domain it is necessary to introduce semantic, machine-understandable, descriptions of Web Services. Using semantic descriptions of Web Services (and of various related aspects like data models or communication patterns) one can build Web Service architectures that solve the interoperability problem to a much higher degree than current systems (or solve it completely, in certain situations).

In this paper we introduce WSMX, our comprehensive execution environment for Semantic Web Services, the reference implementation of the Web Service Modelling Ontology<sup>4</sup>. WSMX is completely built and designed around Semantic Web Services; it internally uses all the concepts that we address. As such, it is an example of a truly semantic service-oriented architecture (SSOA).

The structure of this paper is as follows: first we describe the requirements from the Business-to-Business domain and show why current Web Service standards do not address them by far. Then we introduce the fundamental idea of Semantic Web Services and describe the framework we adopted, namely the Web Service Modelling Ontology. Next we introduce WSMX as the execution environment for Semantic Web Services; we show usage scenarios for WSMX and explain how WSMX addresses the B2B requirements. We describe the architecture of WSMX and how WSMX can be used to realise a very flexible yet robust and reliable Web Service architecture in the sense of a SSOA.

## 2 Requirements for Web Services

Businesses have electronically supported commercial transactions for several decades, although the relevant technologies have changed over time. We focus on the B2B market, as the interoperability problem is larger than in the Business-to-Consumer market: the number of trading part-

<sup>1</sup><http://www.gartner.com>

<sup>2</sup>see <http://www.w3.org/TR/soap/>

<sup>3</sup>see <http://www.w3.org/TR/wsdl/>

<sup>4</sup>see <http://www.wsmo.org/>

ners is higher, the trading volumes are higher and the environment is more heterogeneous, because businesses can influence to a much lesser extent the systems used by their trading partners.

Web Services were specifically developed to address the problem of interoperability between applications [9]. Before the advent of Web Services, architectures of application integration systems used custom adapters to connect legacy applications.

Although Web Services sufficiently address some requirements from the B2B domain, more is needed to solve the integration problem. Several calls for semantic annotation of Web Services have been made, with the following requirements from the B2B domain [2, 3, 4]:

**Discovery:** requesters of certain services need to find the relevant providers of these services. For discovering them one can make use of directory services. The problem with current approaches (for instance UDDI<sup>5</sup>) is that the described functionality of registered services is not machine-understandable, but informal (for instance in natural language). Therefore no mechanised support for discovering services can be given; currently keyword-based search is the only means of finding relevant services.

Discovering services need not necessarily be fully automated (one can find many non-technical objections to fully automated discovery), but support for some richer discovery than keyword-based search is necessary.

**Interoperability:** requesters and providers communicate and exchange data which may lead to interoperability problems. On top of the data mismatch (in structure and meaning) interoperability problems also occur when considering more complex communication patterns (a request-reply is a simple communication pattern, a more complex one could for instance involve a negotiation, and consist of many communicative acts on both sides). In a complex pattern mismatches can occur between the communication pattern of the requester and that of the provider.

Current standards like XML and XML Schema only solve the mismatch on the syntactical and structural level; solving the mismatch on the semantic level is usually handled on a case-by-case basis (for instance using custom adapters). Mismatches on the communication patterns are not dealt with in current standards; semantics of the message exchange sequences is necessary to solve the mismatches on that level.

**Composition:** the service-oriented paradigm builds on the notion of composing virtual components into complex

behaviour. A requester can use the functionality offered by multiple providers without worrying about the underlying differences in hardware, operating systems, programming languages, etc. Each service is designed to satisfy a business task while possibly collaborating with applications or services provided by other entities.

A number of approaches exists for modelling Web Service composition. Although these Web Service composition languages are more suitable than the proprietary languages used in traditional workflow products [17], they lack the possibility to dynamically bind to Web Services at run time [11]. Service requesters have to bind specific services at design time which means they cannot take advantage of the large and constantly changing amount of Web Services available.

**Security and Reliability:** To secure Web Services, a range of security mechanisms are needed to solve problems related to authentication, authorisation, data integrity and data confidentiality. Fundamentally, each message exchange should be private and unmodified between the service requester and service provider as well as non-repudiable.

Traditional network level security mechanisms such as Transport Layer Security and Secure Multipurpose Internet Mail Exchange are the most commonly used point-to-point technologies. They are not sufficient for providing end-to-end security, since Web Services use a message-based approach that enables complex interactions, which can include the routing of messages between and across various trust domains [2].

Web Service security standards are currently emerging and not yet broadly-adopted in Web Service architectures. Security is not specifically related to Semantic Web Services, but crucial for the B2B domain.

### 3 WSMO

Enriching Web Services with semantic mark-up is essential in addressing the previously described requirements. Semantic mark-up can be exploited to automate the tasks of discovering, executing, composing and interoperating services.

The Web Service Modelling Ontology<sup>6</sup> (WSMO) is a formal ontology for describing various aspects related to Semantic Web Services. The objective of WSMO and its surrounding efforts is to define a coherent technology for Semantic Web Services by providing the means for semi-automated discovery, composition and execution of Web Services which are based on logical inference-mechanisms.

<sup>5</sup>see <http://www.uddi.org/>

<sup>6</sup>see <http://www.wsmo.org/2004/d2/v1.0>

WSMO defines four main modelling elements for describing several aspects of Semantic Web Services: ontologies, Web Services, goals and mediators. In what follows, we will describe all these elements, insisting on their importance in reaching a truly Semantic Web Service technology.

As defined in [5], ontologies are formal explicit specifications of shared conceptualisations. In WSMO they represent key elements, having a twofold purpose: firstly they define the information's formal semantics and secondly, they allow to link machine and human terminologies. The WSMO ontologies give meaning to the other elements (Web Services, goals and mediators), and provide common semantics, understandable by all the involved entities (both humans and machines).

In WSMO, requesters of a service express their objectives to be solved by Web Services as goals, which are high level descriptions of concrete tasks. Every requester expresses its goal in terms of its own ontology, which, on one hand provides the means for a human user to understand the goal, and on the other hand allows a machine to interpret it as part of the requester's ontology. Another advantage of using goals is that the requester only has to provide a declarative specification of what it wants, and does not need to have a fixed relation with the Web Service or to browse through an UDDI registry for finding Web Services that provide the appropriate capability.

To accomplish this goal the requester (by means of its information system) has to find an appropriate Web Service, which may fulfil the required task. Similar to the way the requester declares its goal, every Web Service has to declare its capability (that is, what it is able to accomplish) in terms of its own ontology. If the requester of the service and the Web Service that offers it use the same ontology the matching between the goal and the capability can be directly established. Unfortunately, in most of the cases they use different ontologies, and the equivalence between the goal and the capability can be determined only if a third party is consulted for determining the similarities between the two ontologies. Another problem that may appear is the incapacity of the requester and of the provider of the service to communicate with each other, the reason for this being the heterogeneity of their communication protocols. For these reasons, WSMO introduces the fourth key modelling element: the mediators, which have the task of overcoming the heterogeneity problems, both at data level and at communication level.

## 4 WSMX

The Web Service Execution Environment (WSMX)<sup>7</sup> is a reference implementation for WSMO, designed to allow

<sup>7</sup>see <http://sourceforge.net/projects/wsmx/>

dynamic discovery, invocation and composition of Web Services. WSMX offers complete support for interacting with Semantic Web Services. In addition, WSMX supports the interaction with non-WSMO, but classical Web Services ensuring that a seamless interaction with existing Web Services is possible.

WSMX is a useful framework for both Web Service providers and requesters. As a provider, one may register its service using WSMX in order to make it available to the consumers and, as a requester, one can find the Web Services that suits their needs and then invoke them in a transparent, secure and reliable way. WSMX itself is made available as a Web Service, so either for finding a Web Service or for actually invoking Web Services a requester has just to invoke WSMX itself. In the first case, a formal description of requester goal has to be provided, and in the second case, the actual data the requester wants to use for the invocation. In this way, WSMX can take care of all the other required computations such as heterogeneity reconciliation, composition, security or compensation.

WSMX supports a common B2B scenario, acting as an information system representing the central point of a hub-and-spoke architecture. If two partners want to communicate they only abstract their functionality to WSMX. WSMX itself is a Web Service, so if the applications offer their functionality as a WSDL interface no adapters need to be developed. Acting as this central integration point WSMX reduces the total number of interfaces from  $n(n-1)$  to  $2n$ : instead of interfaces to each partner, only single interfaces to WSMX are necessary.

We will start the next section with a brief presentation of the WSMX architecture, followed by a more detailed description of some of the components default implementations.

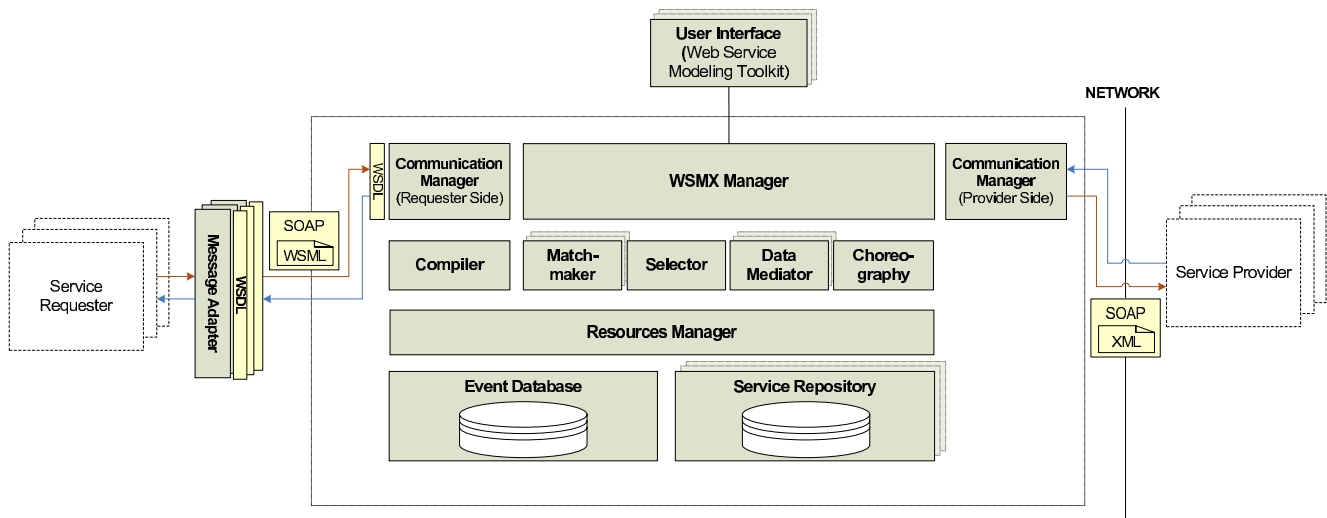
### 4.1 WSMX Architecture

As mentioned WSMX has two operational aspects [12]: the registration and the execution of Semantic Web Services. Figure 1 presents the WSMX architecture and its most important components.

The Registration process (see Figure 2) is used to register descriptions of entities with the WSMX system. The WSMO Editor is used to create the description of the Web Services, ontologies, mediators and goals. These descriptions are passed to the Compiler for validation and for storing the compiled data in the repository.

The Execution process consists of two phases: the Discovery and the Invocation of Web Services. The first phase identifies those Web Services that suit the requester goal, the second phase makes the actual invocation of the selected Web Service.

In the Discovery phase, the Matchmaker matches the ca-



**Figure 1. Simplified WSMX Architecture**

pability of Web Services in the Service Repository against the requester goal and returns a set of Web Services. The Selector component chooses one Web Service that best suits the requester preferences. The Data Mediator can be used to overcome the eventual mismatches that can occur between the goal and capability formalisations.

In the Invocation phase, the Communication Manager invokes the selected Web Services. As in the previous case, the help of the Data Mediator may be required, this time to transform the incoming data from one conceptualisation (used by the requester) into another conceptualisation (used by the provider). Also the Choreography Engine is used to link the communication patterns (choreographies) of the requester and of the Web Service.

The WSMX Manager implements the execution semantics of the system and offers the underlying mechanisms for an event-based architecture. The Resource Manager offers an abstraction layer to the persistence layer.

## 4.2 WSMX components

This section describes in more detail the most important components of the WSMX architecture presented above. WSMX offers default implementations for these components. These are connected to the architecture through a set of placeholders that have precisely defined interfaces; this allows external components to be easily plugged-in. Existing implementations of components can thus be replaced over time with alternative or more expressive implementations. In addition, using the same mechanism, WSMX aims to offer dynamic discovery of Web Services having the same functionality as the default components, in order to achieve higher performance and accuracy.

### 4.2.1 Adapter

Adapters address a problem occurring even before the above mentioned interoperability becomes an issue, namely when trying to connect external systems to WSMX. Although they are outside the WSMX architecture (see Figure 1), we briefly describe them here to emphasise their role in overcoming data representation mismatches on the communication layer. These systems, often referred to as back-end applications, do not natively support WSML and may not be able to directly send messages to a WSMX instance. On a conceptual level, an Adapter transforms the format of a received message or even extracted data from an API into the WSML compliant format understood by WSMX. The transformation based on mapping rules is concerned with the syntactical mapping of the messages formats while maintaining the semantics of the message.

### 4.2.2 Compiler

The Compiler component is responsible for checking the syntactical validity of WSML documents and for storing the parsed information persistently. The Compiler is being used in both operational aspects of WSMX (registration and execution). During the registration of services the Compiler reads a service description, validates and finally stores it. During execution of services the Compiler reads a goal description, validates and stores it, after which the goal is passed to the management component in order to be resolved.

Since various projects are already implementing different systems using WSMO, an initiative has started for an Open-Source WSMO API<sup>8</sup>. This API and its accompa-

<sup>8</sup><http://wsmo4j.sourceforge.net/>

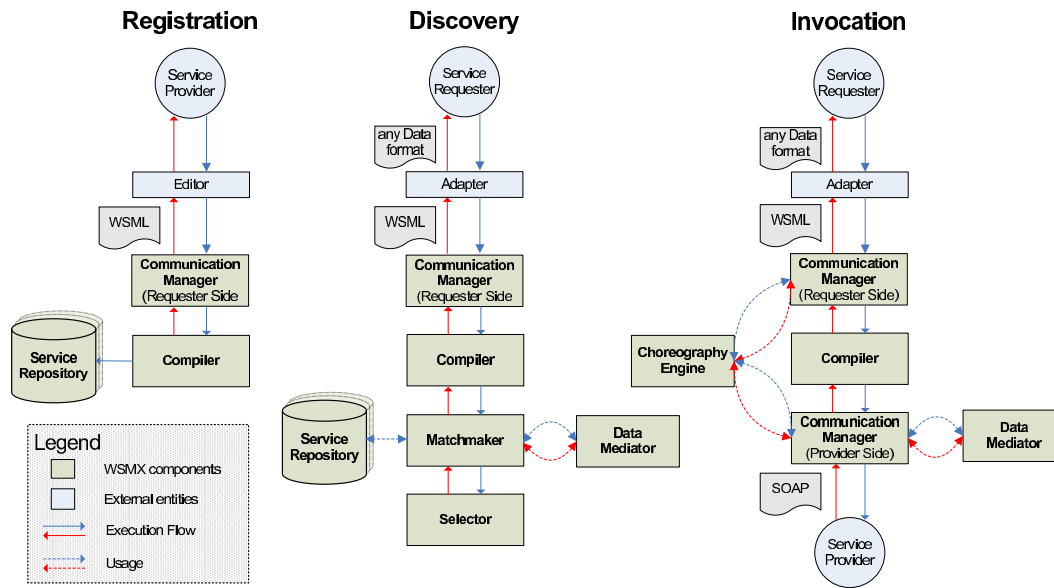


Figure 2. Simplified Operational Aspects of WSMX

nying reference implementation offer methods for parsing a WSMO document and for constructing an in-memory model of this document, for querying and modifying this model and for serialising it into different persistent storage solutions. The work on this API is integrated with the work on the WSMX Compiler component.

#### 4.2.3 Matchmaker

The matchmaking component is responsible for finding appropriate Web Services to achieve a goal. This component's input is a set of (existing) Web Services and a requester's goal; the output is a set of Web Services that can fulfil the goal. A different problem is finding the set of all existing Web Services (discovery); we have not yet addressed this (non-trivial) problem: WSMX contains a local repository of known Web Services and can also use external (UDDI-like) repositories.

As previously mentioned, the capability of a service describes what the service promises to deliver while the goal describes what a service requester wants to have achieved. Matchmaking should come up with those services that can be used to fulfil the goal. The user may be interested also in services that will not exactly deliver what they are asking for, but are to some extent related to it. Therefore, several degrees of matches can be considered, each varying in the degree of satisfaction of the user's goal. The notion of matching goals to services is similar to component match-making, cf. [10].

In the first release of the WSMX implementation, the built-in matchmaking is performed by simple string-based

comparison of the requester goal with the various Web Service capabilities available in the WSMX repository. However, several techniques and implementations [6, 7] are already being developed for doing discovery on WSMO goal and services and are designed to be adopted for WSMX. The event and component based architecture of WSMX makes it possible to adopt implementations for the discovery component, resulting from the ongoing research, once such implementations become stable.

#### 4.2.4 Data Mediator

The Data Mediator corresponds to the ooMediator from the WSMO specifications, being the only type of the four mediators described in WSMO we implemented so far. Thus, the main assumption we make is that both the sender of the data (in our case the service requester) and the receiver of the data (the service provider) use ontologies as means for expressing the semantic of their data. Development and imposing a global ontology is not a realistic nor a feasible approach both from a technical and business point of view. Such a global ontology should be general enough to cover all the partners needs in a consistent way as well as the potential changes of these requirements in dynamic manner. In addition, the trust issues together with the requirements of a competing environment leads to the usage of different ontologies. It is the task of this component to transform the incoming data from the terms of sender's conceptualisation (source ontology) in terms of the target's conceptualisation (target ontology).

The Data Mediator part of the WSMX architecture rep-

resents only a subcomponent of a complete Data Mediation Component<sup>9</sup>. That is, the Data Mediation Component has two main sub-components: a design-time component used for identifying the similarities of two given conceptualisations of a domain, and a run-time component that performs the actual transformations on the working data. The run-time subcomponent is the one present in the WSMX architecture and even if the design-time component is not explicitly part of the WSMX architecture, it enables the second one's functionality, i.e. validated mappings are saved in given storage for being picked up later by the run-time subcomponent.

The run-time subcomponent (our WSMX Data Mediator) is able to load the stored mappings, to create the necessary rules (a rule is a language specific representation of mappings), to apply them to the input data (source instances) and to pass forward the mediated data (target instances). The whole process is executed automatically, using the previously identified similarities. While the design-time component acts on the schema level of the two ontologies, the run-time component acts on instance level, creating the target instances to be used in further computations (e.g. in the invocation).

#### 4.2.5 Choreography Engine

The choreography of a Web Service defines its communication pattern, that is, the way a requester (which may as well be another Web Service) can interact with it. The requester of the service has its own communication pattern and only if the two of them match precisely, a direct communication between the requester and the provider of a service may take place.

In the context of communication pattern heterogeneity equivalence has a different meaning than for the data heterogeneity: by communication process equivalence we understand the full matching of the communication pattern from the source and target parties. That means that for each possible instance of the source choreography at least one instance of the target choreography is available.

Since usually the client has its own communication pattern that in general is 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. In order to communicate the two parties must be able to redefine their communication patterns (or at least one of them has to) or to use an external mediation system as part of the process. The first solution is generally a very expensive one implying changes in the entities' business logic, and it is not suitable in a dynamic environment since every participant would have to readjust its pattern (through reprogramming) each time it gets involved in a new partner-

ship. As a consequence, the role of the mediator system will be to compensate the client's communication pattern or the Web Service's communication pattern in order to obtain an equivalent processes.

The role of the Choreography Engine is to put together the necessary means for the runtime analyses of two given choreography instances and to use the 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. The above presented functionality is based on a design time process which identifies the equivalences between the choreographies' conceptual descriptions, that is, a set of rules are created and stored, in order to be later applied on the particular choreography instances, during runtime.

#### 4.2.6 Composition

The composition component is responsible for executing complex compositions of services in order to achieve a certain goal. The language for specifying these compositions is still under consideration in WSMO. We have investigated two initial approaches, to use hard-coded business rules for composing goals and to embed WSMX in an external process language [14]. Given the component-based architecture of the WSMX system we believe that defining and implementing a specific composition formalism in a later stadium can be achieved without problems.

#### 4.2.7 Communication Manager

The Communication Manager has two major tasks: first, to handle the various invocations that may come from requesters and second, to invoke Web Services and to retrieve the results of these invocations back to WSMX (this could happen either as a consequence of a synchronous call or by a separate invocation of WSMX in case of asynchronous calls).

Currently, even if a semantic description is provided for a certain Web Service capability (e.g. in order to register to WSMX), the actual invocation still has to be made in a classical way, by representing all the data needed for the invocation in XML format. On the other hand, all the WSMX components and all its internal mechanisms operate using the semantic descriptions provided by WSMO. In order to make the bridge between the semantic descriptions and the classical syntactic Web Service descriptions, WSMX provides the necessary means for lifting non-semantic descriptions (e.g. XML messages and XML schemas) to a semantic level and to lower the elements semantically described (e.g. ontology instances and concepts) to the level required by the classical approaches. For accomplishing these tasks,

<sup>9</sup>see <http://www.wsmo.org/2004/d13/d13.3>

additional, intermediary components need to be introduced to perform the lifting and lowering operations, the Adapters (addressed in a previous section).

These two operations described above (i.e. lifting and lowering) may seem as burdensome and not very elegant from the Semantic Web Services point of view. But we believe that this is only an intermediary solution meant to compensate the current lack of fully semantically described Web Services (e.g. as WSMO presents them) on the Internet. For Semantic Web Services described according to WSMO specifications, the Communication Manager task is much simpler: no calls to Adapters are necessary, but only a simple invocation with the proper data as the semantics of the service interface specifies.

## 5 Related Work

Since WSMX is a sample implementation of WSMO, we have to differ between implementations that are also based on WSMO and implementations that use other frameworks.

IRS [13] is the only other execution environment based on WSMO, and can interoperate with WSMX. With the current version of IRS3 a service provider can create a WSMO service description that can be published against their service on the IRS3 server. Once the service description is available, a goal can be described in WSMO and bound to the published Web Service description using a mediator. The main limitation of this approach is that the binding 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.

OWL-S [15] is an OWL-based Web Service ontology, where service descriptions define what the service provides for the seeking agent, describe how the service works and what happens when it is carried out and specify how the service should be used in terms of the communication protocol, the message format etc. The main differences in the conceptual model between WSMO and OWL-S is that WSMO differs between the service requester and the provider and it introduces the concept of mediators to enable the requester of services to use different terminologies than the provider. An extensive comparison of WSMO and OWL-S can be found in [8].

No complete toolset for OWL-S is available, only separate tools such as a composer, a matchmaker and an editor [16]. These tools suffer from the limitations of OWL-S, they especially lack a mediation facility.

METEOR-S [18] works with existing Web Services technologies and combines them with ideas from the Semantic Web to enable Web service discovery and composition. It does not introduce a new ontology language, but uses DAML+OIL and RDF(S) to map WSDL message types (inputs, outputs) and operations to concepts in domain

ontologies. The core tool is called MWSAF, and consists of three components: the ontology-store, the translator library, and the matcher library. Another tool is provided for adding semantics to UDDI registries and Web Services and for discovering them, called METEOR-S WSDI. It uses a specialised ontology to map each registry to a specific domain. WSDL operations are also mapped to concepts from an operation's domain ontology. By annotating their preconditions and effects, they can be matched with a user's goal, expressed using service templates with the operation, inputs, outputs, preconditions and effects.

METEOR-S is limited in the expressivity of the semantic mark-up: it only annotates the inputs and outputs of a Web Service. Secondly, it lacks an independent execution environment, for the actual execution of Web Services an external engine is used [1].

Beside these scientific tools there are several commercial B2B servers available from different vendors. All of the major software vendors offer solutions to address the needs from the B2B domain. The majority of these integration tools has initially been developed before the advent of Semantic Web Service. Hence they lack any semantic annotation of services and do not fully support discovery and interoperability.

## 6 Conclusion and Further Work

Web Services represented a step forward in enabling the collaborations between various entities on the Web and in overcoming the imminent interoperability problems that may appear. Business-to-Business can fully benefit from their usage by allowing business entities to expose their capabilities and to consume the functionality offered by their partners. Therefore, information systems based on a service-oriented architecture able to integrate different functionalities and to offer a virtual component model that abstracts from the peculiarity of specific implementations, seem to be a very appealing solution. In this paper we have described how WSMX tackles the requirements occurring in B2B collaborations. As such WSMX applies a new paradigm: Semantic Web Services. Based on the principles of WSMO, this semantic annotation allows WSMX to address the interoperability and discovery issue in a different manner than currently available integration tools.

WSMX (using WSMO discovery approaches) can find service offers that logically match with the service requested; if terminology differences occur in the descriptions of the service offer and the service request, WSMX will mediate those differences. WSMX can invoke selected services to achieve the request, again mediating between data and processes differences. We have provided a default implementation for such a mediator which makes use of a set of mediation rules specified for the given ontologies. For

the next versions we plan to extend the Choreography Engine functionality by a Process Mediation module, in order to allow the compensation of the potential differences in message exchange patterns of the two business entities.

Using WSMX, a business expert, while trying to capture the process logic in an application, can build richer service compositions. Instead of composing a process from statically linked services, they can now describe their requests at activity points in the composition as goals and WSMX will try and execute a service that matches this goal. We have described our ongoing research in formalising and implementing a composition language and engine inside WSMX; currently we have two initial approaches: firstly to use hard-coded business rules to compose goals, secondly to embed WSMX in an existing process language.

Since the focus of the development of WSMX has been on the architecture and the components that facilitate this new paradigm of Semantic Web Services, we have postponed the integration of a security component that allows authentication, authorisation and communication security until the next release. We do however provide a basic recovery mechanism through our event-based model with persistent storage of all events in the system.

We have tested our system in a real-world use case, involving ordering broadband Internet lines from different telecommunications providers [14]. We concluded that for full support of this use-case we need to extend the WSMO model slightly, and also we need to provide process management support (service composition) in WSMX, which is planned for a future release.

We realise that in this first release of WSMX we have not yet solved all the described problems from the B2B integration domain. However, since we use the new paradigm of Semantic Web Services it is possible to address these problems comprehensively. We already thoroughly address the discovery and interoperability problems; we plan to continue with our work in the next release on security and composition. We expect to be able to easily extend the functionality since on the one hand we have a firm conceptual basis using Semantic Web Services, and on the other hand a firm technical basis using our event-based service-oriented architecture.

## References

- [1] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *Proceedings of IEEE International Conference on Services Computing*, 2004.
- [2] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. W3C Working Note, <http://www.w3.org/TR/ws-arch>, 2004.
- [3] S. Burbeck. The evolution of Web applications into service-oriented components with Web services. <http://www-106.ibm.com/developerworks/webservices/library/ws-tao/>.
- [4] C. Bussler, D. Fensel, and A. Maedche. A conceptual architecture for Semantic Web enabled Web services. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 31(4):24–29, 2002.
- [5] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–229, 1993.
- [6] U. Keller, M. Stollberg, and D. Fensel. WOOGL meet semantic web. In *Proceedings of the Workshop on WSMO Implementations*, 2004.
- [7] M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A logical framework for web service discovery. In *ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, 2004.
- [8] R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of WSMO and OWL-S. In *Proceedings of ECOWS 2004*, pages 254–269, 2004.
- [9] F. Leymann. Web services: Distributed applications without limits -an outline-. In G. Weikum, H. Schöning, and E. Rahm, editors, *Proceedings of BTW 2003: Datenbanksysteme für Business, Technologie und Web*. Springer, 2003.
- [10] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the International Conference on the World Wide Web*, 2003.
- [11] D. J. Mandell and S. A. McIlraith. Adapting BPEL4WS for the semantic web: The bottom-up approach to web service interoperation. In *The SemanticWeb - ISWC 2003*, pages 227–241, 2003.
- [12] M. Moran, M. Zaremba, A. Mocan, and C. Bussler. Using WSMX to bind requester & provider at runtime when executing semantic web services. In *Proceedings of the Workshop on WSMO Implementations*, 2004.
- [13] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A framework and infrastructure for semantic web services. In *The Semantic Web - ISWC 2003*, pages 306–318, 2003.
- [14] E. Oren, A. Wahler, B. Schreder, A. Balaban, M. Zaremba, and M. Zaremba. Demonstrating WSMX – least cost supply management. In *Proceedings of the Workshop on WSMO Implementations*, 2004.
- [15] OWL-S: Semantic markup for web services. <http://www.daml.org/services/owl-s/1.0/owl-s.html>, 2004.
- [16] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara. The DAML-S virtual machine. In *The SemanticWeb - ISWC 2003*, pages 290–305, 2003.
- [17] W. M. P. van der Aalst, M. Dumas, and A. ter Hofstede. Web service composition languages: Old wine in new bottles? In *Proceedings of the 29th Euromicro Conference (EUROMICRO'03)*, pages 298–305, 2003.
- [18] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*, 6(1):17–39, 2005.