



D13.6v0.1 WSMX Use Cases

WSMX Working Draft 5 December 2004

This version:

<http://www.wsmo.org/2004/d13/d13.6/v0.1/20041205/>

Latest version:

<http://www.wsmo.org/2004/d13/d13.6/>

Previous version:

<http://www.wsmo.org/2004/d13/d13.6/v0.1/20041005/>

Editor:

Armin Haller

Author:

Armin Haller

This document is also available in a non-normative [PDF](#) version.

Table of contents

1. Introduction

[1.1. Overview](#)

[1.2. Purpose of this document](#)

[1.3. Document Overview](#)

2. WSMX architecture components

3. Trading Partner Integration

3.1. B2C

[3.1.1. What is B2C](#)

[3.1.2. Use case foundation](#)

[3.1.3. Actors, Roles and Goals](#)

[3.1.4. B2C system architecture](#)

[3.1.5. How WSMX can be utilized](#)

[3.1.6. Advantages WSMX exposes for the B2C use case](#)

3.2. B2B

[3.2.1. What is B2B](#)

[3.2.2. Use case foundation](#)

[3.2.3. Actors, Roles and Goals](#)

[3.2.4. B2B system architecture](#)

[3.2.5. How WSMX can be utilized](#)

3.2.6. Advantages WSMX exposes for the B2C use case

4. Application Integration

4.1. A2A

4.1.1. What is A2A

4.1.2. Use case foundation

4.1.3. Actors, Roles and Goals

4.1.4. A2A system architecture

4.1.5. How WSMX can be utilized

4.1.6. Advantages WSMX exposes for the B2C use case

5. Conclusions and Further Directions

References

Acknowledgement

1. Introduction

1.1 Overview

The Web Services Execution Environment (WSMX) [Oren et al., 2004a] is an execution environment for dynamic discovery, selection, mediation and invocation of semantic Web Services. WSMX is based on the Web Services Modeling Ontology (WSMO) [Roman et al., 2004] which describes all aspects related to this discovery, mediation, selection and invocation. WSMX is a reference implementation for WSMO. The goal is to provide both a test bed for WSMO and to demonstrate the viability of using WSMO as a means to achieve dynamic inter-operation of semantic Web Services.

1.2. Purpose of this document

This document exemplifies several usage scenarios of WSMX. In the current version of this document we identify three possible use cases, namely a Business-to-Consumer (B2C), a Business-to-Business (B2B) and an Application-to-Application (A2A) scenario, and showcase how WSMX can be utilized for them. The process steps within these scenarios are only presented at a conceptual level, a more detailed description of already implemented parts of the execution semantics of WSMX is given in [Oren, 2004].

We briefly comprehend the architecture and the scope of WSMX and give a review on related knowledge necessary for exemplifying the specific use cases. The current architecture, represented in a preliminary framework, is aligned to WSMO Standard [Roman et al., 2004]).

The first version of WSMX provides a complete architecture for dynamic discovery, mediation, selection and invocation and a simple but complete implementation of these components. In the current status it is not applicable to fulfill all aspects of the execution requirements described in these use cases. Subsequent versions of WSMX will incorporate the ongoing research of the WSMO and WMSL working group and will lead WSMX to a status where it fulfills all the Semantic Web Services support described in this document.

Hence this deliverable is intended to serve as an input for further improvements and providing valuable insight for adapting or including components to deal with real world scenarios for Web Services. On the other hand this deliverable will evolve in accordance to the ongoing development of WSMX itself and further use cases will be included in the future if necessary.

1.3. Document Overview

The use cases we will carry out are divided into two business domains. The B2C and the B2B use case are abstracted to Trading Partner integration. Both deal with the same purchasing process, but with different actors and roles. Application Integration contrariwise exemplifies a use case within a company which raises the demand for a so called intra-Enterprise integration [Bussler, 2003].

The document is organized as follows. [Section 2](#) briefly describes the WSMX architecture components and in particular figures out which components will be addressed during the use cases. [Section 3](#) describes the purchase of a bike, firstly in a B2C and secondly in a B2B collaboration. Both scenarios are first described in general, whereas the actors, roles and goals are defined, and then the overall system architecture is depicted and finally the execution flow of WSMX is outlined. [Section 4](#) describes an Application Integration scenario by illustrating an A2A use case. It depicts an automatically invoked stock purchase by an application running in a banks division and processed and executed by another division within the same bank. [Section 5](#) finally concludes the document.

2. WSMX architecture components

The following chapter gives only a very brief description of the components. This is solely indented to give the reader the chance to follow the process steps in the use cases without knowledge of other WSMX deliverables. Given the fact that some components are very specific and cover some technical requirements, they are not exemplified in the use case and therefore not specified in this chapter. For further information on all components we point the interested reader to the following deliverables [Oren et al., 2004] [Moran/Zaremba, 2004].

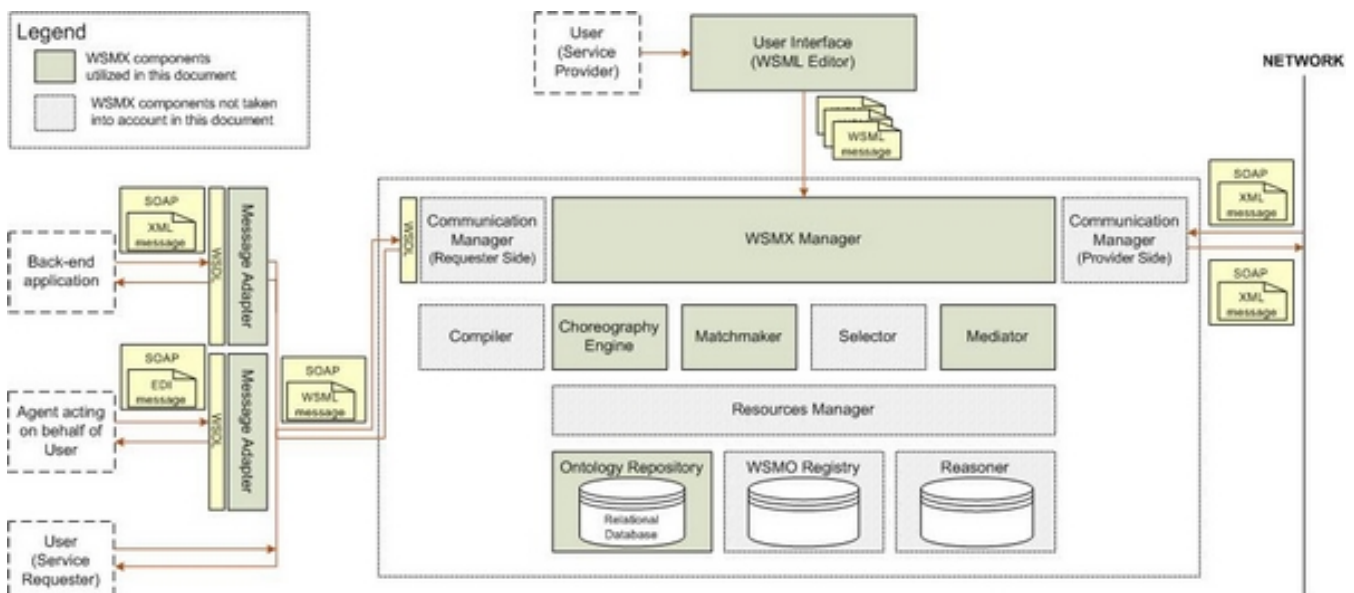


Figure 1: WSMX Architecture ([click to enlarge](#))

- **Message Adapters:** These components are in fact external to the WSMX architecture, but as long as back end applications do not offer a WSMO API they are essential for connecting any kind of back-end application to the WSMX server. The *Message Adapters* allow transforming from any message format (e.g. RosettaNet, UBL, EDIFACT, ANSI X12, xCBL etc.) to the expected WSML message format.

- **User Interface (WSML Editor):** This component is used by the service provider to create the description of the Web Services, Ontologies, Mediators and Goals.
- **Communication Manager:** The *Communication Manager* has a twofold purpose. On the one hand it provides an interface to the *Adapters* to accept and send WSML messages and on the other hand it is responsible for translating the message into whatever data representation is required by the external entity to be invoked. This translation is often referred to as 'lowering'. To do so the Communication Manager interprets the interface part of the WSMO service description to determine which binding is necessary for the service. Where it is not fixed, the transport protocol and associated security required for communicating with the external entity may also be selected. After invocation the *Communication Manager* is responsible to translate the message received from an external entity using any data representation to WSML. This translation is often called 'lifting'.
- **WSMX Manager:** This component is the coordinator within the architecture. Although it is not an essential component to describe the use cases, it is mentioned here because it denotes the Service Oriented Architecture (SOA) approach of WSMX. All data handled inside WSMX is internally represented as a notification with a type and state. The *WSMX manager* manages the processing of all notifications by passing them to other components as appropriate.
- **Ontology Repository:** WSMX will offer the management of capability descriptions stored in a repository. This repository could be centralized or decentralized, whereas this use case and the current architecture only scopes with a decentralized repository in each WSMX. These repositories are designed to store, search, retrieve and manage WSMO descriptions of Semantic Web Services. Within this document the name *Capability Repository* is synonymously used for *Ontology Repository*.
- **Matchmaker:** WSMX will offer a set of usable Semantic Web Services by matching capabilities stored in the repository with the goal provided by the user. In subsequent versions WSMX will even be capable to fulfill goals by a composition of the capabilities of several Semantic Web Services. In both cases the result of the *Matchmaker* can be zero, one or many Web Services.
- **Mediator:** WSMX will offer mediation of communicated data. The mediation component tries to determine a *Mediator* for a request in case this is necessary. This mediation can be between two or more ontologies in the matchmaking process and the opposite way after invocation to mediate between the instance data of a known ontology provided by the executed Web Service to the required data in the invoking application. Another application of *Mediators* would be the mapping between the data provided in the input of the goal to the actual required input of the Web Service.
- **Choreography Engine:** The choreography of a Web Service defines its communication pattern, that is, the way a requester can interact with it. The requestor of the service has its own communication pattern and only if the two of them match precisely, a direct communication between the requestor and the provider of a service may take place. Since the clients communication pattern is in general different from the one used by the Web Service, the two of them will not be able to directly communicate, even if they are able to understand the same data formats. The role of the *Choreography Engine* is to mediate between the the requester's and the providers's communication patterns. This means to provide the necessary means for a runtime analyses of two given choreography instances and to use *Mediators* to compensate the possible mismatches that may appear, for instance, to generate dummy acknowledgement messages, to group several messages in a single one, to change their order or even to remove some of the messages in order to facilitate the communication between the two parties.

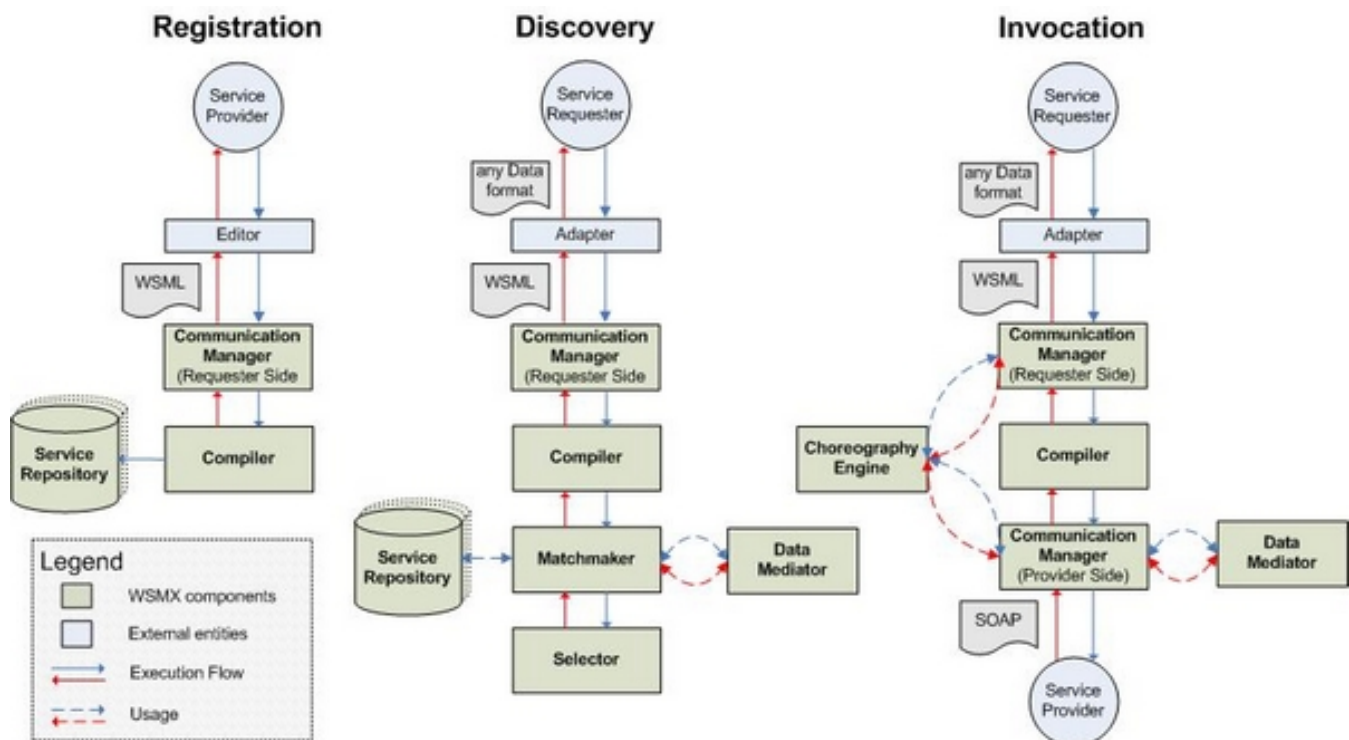


Figure 2: Simplified Operational Aspects of WSMX ([click to enlarge](#))

WSMX has two functional aspects, the *Registration* at design time and the *Execution* at run time. The *Registration* process (see Figure 2) is enabled mainly by the use of the WSMO Editor and the Compiler. In our use cases we imply that the *Registration* of Web Services to WSMX has already taken place.

The *Execution* process has two phases as depicted in Figure 2: the *Discovery* and the *Invocation* of Web Services. The first phase has the role of identifying those Web Services that suit the requester's goal, while the second phase makes the actual invocation of the selected Web Service. A detailed description of the execution flow is given in the respective use cases.

3. Trading Partner Integration

The following use cases are high level descriptions of business collaboration. Hence we currently ignore issues like communication security, transaction management, error handling and compensation. Furthermore we imply in these scenarios that all participants, exposing business functionality as a Web Service, agree to fulfill the real world interactions regardless of the respective business partner invoking them. Hence we imply that business partners trust each other, although they are maybe unknown to each other and do not have agreed on Service Level Agreements (SLA) beforehand. We will take this into consideration in further versions of this document as well as in improvements in the WSMX architecture. It is currently planned that SLAs between two partners will be taken into consideration in the non-functional properties of the goal and capability description.

The choice to describe the purchase of bicycles in the following use cases has been made deliberately, because they represent non-commodity goods. A commodity good is traded primarily on the basis of price and not on differences in quality or features. Manufactured goods are said to be commodity goods if purchasing decisions are made almost solely on the price of the product. By choosing the bicycle as a non-commodity good we obviously have to deal with selection criterias beyond the price.

3.1. B2C

3.1.1. What is B2C

B2C stands for "business-to-consumer" and applies to any business or organization that sells its products or services to consumers. The most commonly known B2C E-Commerce market participant is the book retailer Amazon.com, which launched its website in 1995. However, in addition to solely online based retailers, also bricks-and-mortar companies have included many B2C services on their websites, such as online banking, travel services, auctions, betting and accommodation search.

3.1.2. Use case foundation

Let us imagine a Bicycle retailer, further called BCR, which provides the disposal of several kind of bikes, including mountain-, racing-, city- and juvenile ones over internet. The following scenario is not depicted by using a traditional retailer with bricks-and-mortar branches, a fixed number of business relations with manufacturers and wholesalers and tightly coupled applications, but with one that tries to enhance its online business model by applying Semantic Web Services. By utilizing WSMX as the means to handle them, the BCR can enlarge its formerly limited number of bicycle types and brands offered for sale. In some respect the BCR migrates from a traditional retailer to a provider of a central knowledge repository for bicycle purchases. The operation of real world branches wouldn't have any influence on the use case itself. Naturally this kind of business model does not necessarily have to be offered by a bicycle retailer. It can be any company provideing a Web Service repository also covering the bicycle domain. To simplify matters and in respect to basic economic effects of specialization we assume that a retailer will offer this supplementary functionality prior to others.

Fundamentally we can differ between two service models the BCR can offer to its customer.

1. The first one is coherent with current practice in E-Commerce. The BCR runs a **portal website**, but unlike ordinary B2C portals, this website differs by utilizing Semantic Web Services. The user is not only able to search for different bicycles from different vendors and browse through the resulting offers to decide, if one fits to their needs, but they are also able to provide the BCR a specific goal in formal semantics. How the user can provide the goal via the portal is discussed in [Section 3.1.4](#). The goal represents the user's tangible needs, based on which WSMX will find Web Services capable of fulfilling the user's goal.
2. The second one would be that the BCR only **hosts a WSMX server** and the customer uses a client side version of a WSMX editor. This editor allows the user to define a goal and connect to a WSMX server afterwards. WSMX tries to match this user goal with Web Services offering the desired service.

Excursus: In the latter service model the BCR could in fact become dispensable. WSMX will be able to connect to capability repositories of other WSMX's to search for their capability store. This means for our example, if all bicycle vendors utilize WSMX as their Web Service execution environment or at least provide a WSMO compliant description of their Web Service in any WSMX or central repository, the customer's client goal editor would be able to find this Web Service by connecting to any arbitrary WSMX. For further descriptions of our use cases we assume the existence of a BCR which hosts capability descriptions of several services from

different manufacturers. This will be the most likely scenario in the near future, as far as a central repository of Semantic Web Services (like Google is for websites) or a central trading platform for semantic Web Services (like Ebay is for consumer goods) is not established so far.

3.1.3. Actors, Roles and Goals

Within the B2C view we focus on two different actors. The following listing defines in detail why they are interacting (goal), with whom and in what way (role) and what assumptions are essential.

- **Customer:**
 - *Goal:* buys a city-bike
 - *Role:* the customer interacts with the BCR for service usage, payment and acceptance of the bike in the requesting role
 - *Assumption:* either uses a WSMX editor or utilize the functionality offered by the portal website of the bicycle retailer
- **BCR:**
 - *Goal:* provides services to customers by aggregating the product range of different bike manufacturers; maximizes profit by selling as many bikes as possible at a profitable price.
 - *Role:* acts as an intermediate between the customers and different bicycle manufacturers interacting with its customers via a user interface (online based portal) or by providing a WSMX repository accessible over internet.
 - *Assumption:* either provide web portal allowing customers expressing their goals or running WSMX as a stand-alone server for direct access by the customers editors.

3.1.4. B2C system architecture

Since the BCR can offer its service in at least one of the two above mentioned ways and the user's goal has to be described in WSMX [Oren et al., 2004b], we have to differentiate between two system architectures reflecting the two service models described above. Figure 2 depicts both architecture types, whereas the manufacturer is not relevant for the B2C use case. The interaction between the BCR and the manufacturer is described in the B2B use case.

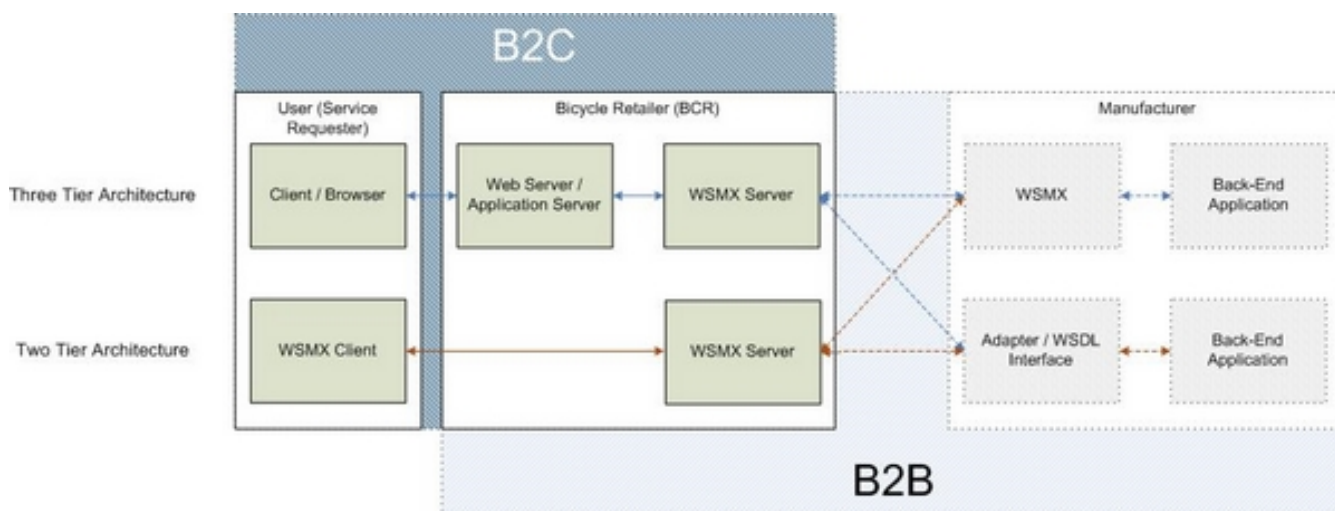


Figure 3: B2C architecture types (click to enlarge)

Three Tier Architecture: Client Browser - Web Server/Application Server - WSMX Server (see Figure 3)

This architecture type reflects the requirements of the first service model in 3.1.2. Within this architecture we can differ between two possible implementations of the portal site in the middle tier:

- The first and more unlikely case is that the retailer only provides the means to input the goal directly in WSML. Practically we can rule out this solution, since the end user (customer) should not be bothered with expressing their goal directly in a formal language. This might be applicable as a second choice for experienced users, but not as the only means for describing a goal.
- The second possibility would be to choose out of a range of predefined goals by only changing the postConditions and effects patterns. The user would simply have to change some variables in the hidden logical description of the goal according to their requirements. This can be either goals provided by the BCR retailer, fed to their own WSMX repository or goals already provided in the release version of the WSMX repository itself. Both have to be made accessible on the portal website.

Two Tier Architecture: WSMX Client Editor - WSMX Server (see Figure 3)

This architecture type reflects the requirements of the second service model in 3.1.2. The users can define their goals by using a GUI based client side WSMX editor. This editor provides the means to graphically represent logical expressions in a way that the user only has to deal with some kind of symbols by changing their properties and relationships to each other.

3.1.5. How WSMX can be utilized

The following paragraph describes the interactions between the user and the BCR retailer in more detail and specifies which specific component in WSMX is utilized to perform the required task. The way how the interaction between the BCR and the manufacturer take place are not presented in this use case, since they are discussed in detail in the following B2B scenario. The above mentioned customer has a goal to buy a bicycle. Let us assume that this customer is an average skilled biker without any preference regarding the manufacturer of the bike. Their only conditions on the effects in the sense of WSMO are as follows:

- Buy a city bike,
 - with a price less than 400 €,
 - with dirt deflectors at the front and rear tires,
 - and with at least 21 gears.
1. The first step is that the customer expresses this goal in one of the above mentioned ways:
 - The BCR offers predefined goals, out of which the user can choose one to customize it to their specific needs. This could be realized by a specially designed portal website allowing to express logical terms in a user friendly way.
 - Another solution would be a client side editor tool connecting remotely to any known WSMX server.
 2. The message containing the goal described in WSML is sent to the WSMX server. The way this message is sent, depends on where the user defined their goal. This can be in the first case the web server invoking a CGI script generating the necessary WSML message or in the second case the client side WSMX editor directly invoking the API of the WSMX server running at the BCR.
 3. When the new message containing a requester's goal (*Invocation* phase, see [Section 2](#)) arrives, it is picked up by the *Communication Manager* and translated into an internal

representation using the Compiler.

4. The *WSMX manager* now invokes the *Matchmaker* component to find all the Web Services that provide the capability to satisfy the user's goal. In the current specification the matching component just selects all the capabilities that are syntactically identical to the goal; further developments will take logical reasoning into account and therefore provide the means to find as many matching Web Services as possible [Keller et al., 2004]. For Web Services that use a different ontology than the one used by the goal, mediation is needed. Therefore there can be an interaction between the *Matchmaker* and the *Mediator* component, which will also be ignored in this high level use case. The selector component would now be responsible to pick one specific Web Service out of the result set on basis which fulfills the user's goal best based on the non-functional properties and the user preferences. In this B2C scenario and in case of a non-commodity good like the bicycle, this selection couldn't be automated, since a user for example would not be able to describe their preferred bike design in logical terms.
5. The *Communication Manager* generates a WSMML message containing the capability descriptions of each found Web Service, rated and ordered by measuring the non-functional properties and taking the user preferences into account. How this information is displayed at the user interface (either the Client's Browser or in the Client side WSMML Editor) is in the responsibility of the transformation process either in the CGI script acting as an adapter or in the *Adapter* of the Client side WSMML Editor.
6. The user selects the desired Web Service, a new goal including the instance data is created and the message is again passed through the *Adapter* to the *Communication Manager* of the WSMML server.
7. Starting the second phase of the *Invocation* aspect (see [Section 2](#)) the *Communication Manager* first has to invoke the *Choreography Engine* to check if the incoming message is part of an ongoing collaboration. For simplification let us assume that the communication pattern of the requester and the provider in our use case are symmetric, more precisely represent a request-reply pattern, the *Choreography Engine* will respond to the *Communication Manager* that the communication can be continued with the invocation of the Web Service. If the communication pattern of the provider is more complex than the mentioned request-reply style than it is in the task of the *Choreography Engine* to iterate through the message exchange between the requester and WSMML itself to gather the necessary data to invoke the Web Service.
8. The *Communication Manager* is responsible for translating the WSMML message into whatever data representation is required by the external entity and sending it to the respective Web Service.
9. After invocation the *Communication Manager* picks up the result performs the lifting back to a WSMML format and passes the message back to the *Adapter* on the requester side, again either the CGI script or the client side WSMML editor.

3.1.6. Advantages WSMML exposes for the B2C use case

Advantages for the customer: The customer can abandon a protracted search through several on- or offline catalogues to find and then buy bicycles fulfilling their goal. WSMML installed, configured and maintained at the BCR will present them only Web Services exactly offering the possibility to order the desired bicycle. If exactly the same bike is offered by different providers the user can choose the provider with the cheapest price. Hence the customer can be sure that there is no provider on the market (at least one who exposes their Web Service capability in a WSMML repository) who offers the bicycle at a cheaper price.

Another advantage is the one-step ordering made possible by WSMML. After the discovery of

Web Services offering bikes matching their goal the customer can select the service to be invoked by WSMX. Traditional approaches would include the search in repositories (websites, offline catalogues, forums etc.) without the possibility to directly order the bike. By employing WSMX the customer would only find sources where an ordering is possible in the same step as the discovery.

Advantages for the BCR: The advantages for the BCR are even more manifold. First the BCR could avoid a tight coupling of their end user shopping portal (either the web portal or the *Capability Repository* used by the WSMX client side editor) to specific manufacturers/wholesalers. Under the above mentioned assumption that the providers of business functionality via a Web Service fulfill the real world transactions independently who invokes it (without having a SLA), this can be even one, the BCR was not aware of before. Furthermore the customer or in fact the BCR (as mentioned above they provide the means to describe a goal in logical terms) can use different ontologies for describing goals than the service providers use for describing the capability of their service. WSMX will even match the goal with capabilities using different ontologies as long as there is a *Mediator* registered in WSMX to map between the two ontologies. Again this allows a loose coupling and the BCR does not have to use predetermined ontologies impressed by a strong wholesaler or manufacturer.

3.2. B2B

3.2.1. What is B2B

Business-to-Business (B2B) Commerce and B2C Commerce fundamentally differ only in one thing. The customers are different - B2B customers are other companies while B2C customers are individuals. Overall, B2B transactions are more complex and have higher security needs. Beyond that, there are two major distinctions:

- **B2B sales require negotiation**
Unlike B2C scenarios, where a company only has to publish its catalog online, selling to another business is much more sophisticated and involves negotiation over prices, delivery and product specifications.
- **Integration**
Automation is the main advantage of online B2B. Therefore companies selling to businesses have to integrate their systems or provide a standardized interface to allow communication with their customers without human intervention.

3.2.2. Use case foundation

As mentioned above we ignore the negotiation between two business partners and imply that everyone can use the Web Service with the same conditions. But the second differential between B2C and B2B, "Integration", changes the execution flow between the participating parties, namely the BCR and the manufacturing companies, in comparison to the flow between the customer and the BCR fundamentally. Since applications are interacting with each other and any user interaction should be avoided, the matchmaking process and the selection and invocation have to comply to other requirements.

In this scenario we will change the role of the Bicycle retailer from the service oriented view in the B2C use case, where they only act as an intermediary party to the more common role of a Bicycle retailer as a dealer with warehousing. In contrast to the B2C use case it does not matter,

if the BCR only sell their bikes traditionally offline or also offer them online, either as a Semantic Web Service or only via an ordinary online portal.

3.2.3. Actors, Roles and Goals

Within the B2B view we focus on the following two actors. The listing defines why they are interacting (goal), with whom and in what way (role) and what assumptions are essential.

- **Bicycle retailer:**
 - *Goal:* provides high-quality services to customers by offering them a wide variety of bicycles; maximizes profit by selling as many bikes as possible at a profitable price.
 - *Role:* acts as a retailer with bricks-and-mortar branches; consults customers in its branches, (optionally provides an online based portal or a WSMX repository accessible over the internet - see [Section 3.1](#))
 - *Assumption:* BCR has either WSMX installed in its system architecture or ensures a tight coupling of its back end applications with WSMX *Adapters* provided by manufacturers.
- **Manufacturer:**
 - *Goal:* a commercial company that produces different kind of bicycles; tries to maximize profit by selling as many bicycles at or more likely above break even price as possible.
 - *Role:* provides purchasing service to other companies.
 - *Assumption:* exposes service either as WSDL compliant Web Service or employs WSMX to provide its back-end applications functionality as Web Service

3.2.4. B2B system architecture

For the fact that a bicycle retailer is at the far end of a supply chain their business relations are normally mainly based on pull interactions. This means they rather invoke services via interfaces at their closest business partner (e.g. wholesaler, manufacturer) than to expose services to them. Hence, except in sophisticated supply chain management (SCM) scenarios, bicycle retailers have not to offer these interfaces to their business applications e.g. for automatic purchase ordering, delivery inquiries etc. In excluding these SCM scenarios and assuming that the manufacturer has WSMX installed by any means (normally manufacturers expose several Web Services, hence it is easier to register them once in their own repository) we have to consider two architecture designs for our BCR.

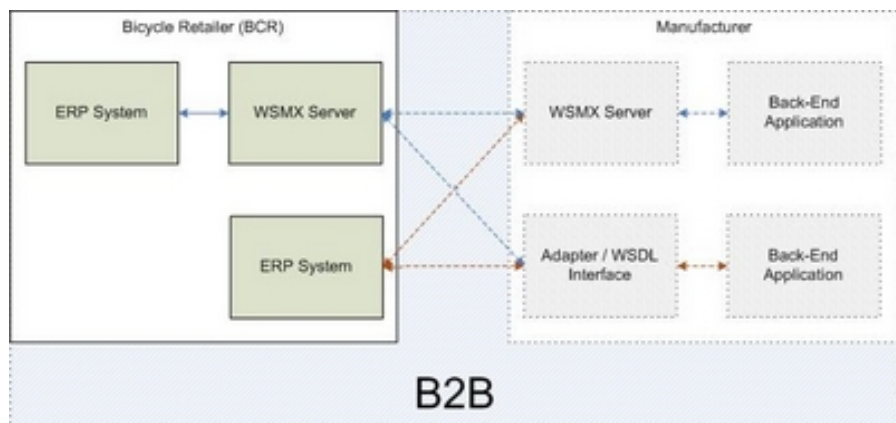


Figure 4: B2B architecture types ([click to enlarge](#))

Architecture with BCR not using WSMX: (see Figure 4)

If the retailer do not expose any Web Service (i.e. only uses services offered by wholesalers and manufacturers) and do not participate in any automated Supply Chain, there is no explicit need to have WSMX installed. By omitting WSMX they would obviously lose its benefits for internal integration as described in [Section 4](#) and they would have to tightly couple their ERP-system with different WSMX *Adapters* of different manufacturers.

Architecture with BCR using WSMX: (see Figure 4)

If the BCR do offer their services as a Web Service to their end customers or business partners, act as an intermediary role like in the B2C use case or want to profit from the advantages WSMX provides in A2A scenarios they would have to run a WSMX server.

3.2.5. How WSMX can be utilized

Since the two above mentioned system architectures differ fundamentally, we have to describe them separately. Starting with the first architecture (no WSMX installed at the BCR) the use case looks as follows.

1. To best serve their customers needs and to have the broadest range of bicycles available in stock, the BCR operates an inventory management system with automatic inventory replenishment. Let us assume that the BCR operates an SAP R3 system with the Material Management (MM) module installed. The inventory management system now initiates a purchase order every time the stock of inventory under-run the recorder point. Because the BCR retailer has not installed WSMX, every purchase in SAP R3 is directly linked to the WSMX *Adapter* interface of a certain manufacturer or wholesaler, which sells the specific bike. This means SAP R3 sends a message containing the goal to purchase a specific bike to the default linked WSMX.
2. The *Communication Manager* of the WSMX server installed as the online front-end at the manufacturer's system architecture receives the message and acts as an event-driven message delivery system based on the goal defined in the received message. Since the BCR does not employ WSMX they have to tightly couple their own SAP R3 system with a WSMX *Adapter* provided by the manufacturer.
3. After parsing and storage the *Matchmaker* picks up the message containing logical expressions and tries to find a matching capability. Due to the tight coupling between the two parties and the a priori agreement which capabilities the manufacturer can offer, the *Matchmaker* will probably find at least one service fulfilling the BCR goal. In most cases it will even find only one Web Service fulfilling its goal, because one functionality within a company is (or should) usually only be offered by one application.
4. Similar to the B2C use case the *Communication Manager* generates a WSML message containing the capability descriptions of each found Web Service and sends it back to the *Adapter* to the ERP system.
5. If only one Web Service capability is returned the system simply sends the goal including the instance data and the message again and it is passed through the *Adapter* to the *Communication Manager* of the WSMX server. If a selection process has to take place it either can be done in the ERP system by predefined preferences or a user interaction has to take place. Another possibility would be to use the selection process of WSMX, but since the system is maintained and operated by the manufacturer, the BCR would not have the rights to define their selection criterias in this system.
6. The two remaining steps, the invocation and the return of the results conform to step number 7 and 8 in the B2C use case. (see [Section 3.1.5](#))

In the second case - having WSMX installed in the BCR's and the manufacturer's system architecture - we can exploit the full potential of WSMX in B2B scenarios.

1. Again we assume that the BCR operates an SAP R3 system with the Material Management (MM) module installed; but this time the ERP system is not linked to the WSMX *Adapter* interface of a certain manufacturer, it is connected to an *Adapter* to a WSMX installed at the BCR itself. This means SAP R3 sends a message containing the goal to purchase a specific bike to the BCR's own WSMX.
2. The message is picked up by the *Communication Manager* and translated into an internal representation using the Compiler.
3. The *Matchmaker* tries to find a capability stored in its own WSMX repository. This process would be comparable to the tight coupling to specific manufacturers mentioned above, since the repository is maintained by the BCR and only known Web Services can be stored. But there are two main differences.
 - There can be multiple results of the matchmaking process since the BCR can have included various Web Services capable of fulfilling the same goal. This means for example that the BCR always uses the cheapest offer to buy the ordered bike, as long as they specify this condition in their preferences. In the tight coupling scenario above the bike would always be purchased from the same manufacturer, even if for example a wholesaler offers the bike cheaper because they want to empty their stocks.
 - The even more important distinction is that the BCR's WSMX will remotely consult other WSMX's, if it is not able to find matching Web Services in its own repository or the retrieved number of Web Service is smaller than defined in the preferences or in the WSMX settings. This has mainly one big advantage. If the BCR has deleted one of the capabilities offered by a specific manufacturer or wholesaler (bankruptcy, issues led to a cancellation of the business connection, etc.) their SAP system is probably still be able to proceed a automatic replenishment order. The WSMX *Matchmaker* will perform the process of finding capabilities matching the goal not only by matching the desired and provided post-conditions and effects of the Web Service but also by taking into account the provided inputs and match them with the pre-conditions of the Web Service. This is crucial to facilitate an automated execution of the whole process; otherwise WSMX would have to ask for additional inputs and an undesired user interaction would follow.
4. The *Matchmaker* will either find none, one or many Web Services. Now either the result set of service descriptions matching the goal are returned to the *Adapter* of the ERP system or the selection process of WSMX selects a service based on non-functional properties and preferences configured in WSMX at design time.
5. The remaining two steps, the invocation and the return of the results again conform to step number 7 and 8 in the B2C use case. (see [Section 3.1.5](#)) Only if the execution of the Web Service was achieved remotely by another WSMX the process would change slightly. The input would be picked up by the *Communication Manager* of the remotely invoked WSMX, lifted to a WSML conform representation and send to the *Adapter* of the initiating WSMX.

3.2.6. Advantages WSMX exposes for the B2B use case

Again we have to break down the advantages regarding the chosen architecture model. Starting with the first one the BCR obviously would not exploit the functionality of WSMX by coupling the BCR's SAP R3 system tightly with one WSMX. Hence there only arise **advantages**

for the manufacturer:

- Firstly not any retailer will tightly couple their systems with one manufacturer. Therefore the installation of WSMX supports, under the premise of a registration of its Web Service capabilities in its own repository, the automatic discovery of the capabilities by any other WSMX.
- Secondly the manufacturer is not bound to provide the functionality by one specific Web Service. If they intend to change the back-end application, the Web Service would still be found, as long as the *Capability Repository* is updated. Obviously the BCR still would have to change the configuration of their ERP system, since the instance data which has to be provided may change, but due to the formal specification this is possible without the manufacturers assistance.

In the second architecture model the advantages for both parties are manifold, but apparently for the manufacturer they are overlapping with the currently described ones. Hence we will only present the advantages now arising for the BCR:

Advantages for the BCR:

- The BCR avoids tight coupling of their back-end application with a specific manufacturers/wholesalers system. WSMX dynamically finds a Web Service in other WSMX's to purchase from this providers. The same as in the B2C case applies here; under the assumption that Web Service providers agree to fulfill the real world transactions independently who invokes the service, this can even be one provider the BCR was not aware of before.
- Similar to the advantages in the B2C use case, the BCR can use different ontologies for describing their goals as the manufacturer to describe the ir capability. WSMX will match with these service descriptions as long as a *Mediator* to map between the two onotologies is available. Again this allows a looser coupling and the BCR has not to use predetermined ontologies impressed by a strong wholesaler or manufacturer.
- By using WSMX the BCR is not only able to find capabilities of other companies, it helps them also to expose their own Web Service functionalities. Hence it would become possibly for the BCR to advertise sophisticated SCM related Web Services (like the sales figures of one specific bike in a period, the type of the most popular bike etc.) in their own WSMX *Capability Repository* which are consequently traceable and executable for downstream companies.

4. Application Integration

Application integration is a buzzword that has been widely used in the past several years, but what exactly does it describe? Enterprises use different back-end applications that need to exchange data. Since they are in most cases originally not designed to exchange data, the need to integrate them arises. To avoid manual intervention to transfer data into one system, which is stored in another one, software technology is needed in order to transfer this data automatically. The following use cases should exemplify on a very high level how WSMX can be used as an integration tool. Technical issues like the particular implementation of *Adapters*, the type of message, which communication channels are used etc. are not in the scope of this document.

4.1. A2A

4.1.1. What is A2A

In contrast to the B2B integration above, application-to-application integration is a subset of B2B integration and therefore the foundation for successful interactions between two companies in a B2B scenario [Bussler, 2003].

Bussler defines A2A as an integration which refers to the integration of any number of back-end application systems that are hosted and require integration. This includes the integration of back-end application systems over a network that resides in a remote division of the enterprise.

Therefore A2A is every integration of back-end applications where the boundaries of the company as a legal entity are not crossed. WSMX will act in this scenario as a system commonly referred to as an Enterprise Application Integration system. Although WSMX will not offer business process management it still covers the two main functionalities which initially made up the foundation to be called an EAI tool [Stonebraker, 2002]:

- message transformation
- adapters

As mentioned WSMX currently does not exploit the functionality of a business process engine, which is referred as compulsory to be called an EAI system by some authors [Linthicum, 1999] [Ruh et al., 2000]. WSMX will offer a Business Process Management (BPM) engine in further versions. Additionally WSMX will offer goal decomposition. This means that the execution of a Web Service by matching an agent's goal would also succeed if the capability is not matched by one singular Web Service, but has to be composed by the usage of several ones. For the agent there is no apparent difference if their goal is fulfilled by one or more Web Services (if it is a single action or a composed one).

Nevertheless the required orchestration and choreography will be part of future WSMX implementations. As soon as these issues are addressed in WSMO and in the architecture we will also include it in this use case scenarios.

4.1.2. Use case foundation

For our application integration use case we will exemplify the purchase of stocks within a bank operating in several countries with different departments dealing with investment funds. A division managing a global hedge fund, further called HFD (hedge fund division), wants to buy stocks. Typically purchases of stocks on different markets within a bank are managed by so called Settlement divisions. In large-scale banks there might be not only one Settlement, but several ones in different countries with overlapping functionality. Similarly the different funds are managed in different departments (e.g. pension funds in different departments than hedge funds), either by outsourced sub companies or independently entities within the bank.

4.1.3. Actors, Roles and Goals

Within the A2A integration we focus on the two following actors. The following listing defines the actors, why they are interacting (goal), with whom and in what way (role) and what assumptions are essential.

- **HFD:**
 - *Goal:* tries to maximize the return of the committed money by lowering its transactional

costs and very simplified by trying to be long in securities outperforming the market and be short in the ones underperforming the market.

- *Role*: provides the management of a variety of hedge funds.

- *Assumption*: WSMX is used by every application in this division to communicate with other applications of other divisions within the bank.

- **Settlement division:**

- *Goal*: provide reliable and high-quality services; minimize transaction costs.

- *Role*: Provides high-quality bank internal transaction management to carry out purchase and sale orders on different security markets; provides services as Web Service as well as via offline channels.

- *Assumption*: WSMX is installed in the settlements system architecture and every application exposes its services via Web Services. Furthermore these Web Services are registered in the *Capability Repository* of WSMX.

4.1.4. A2A system architecture

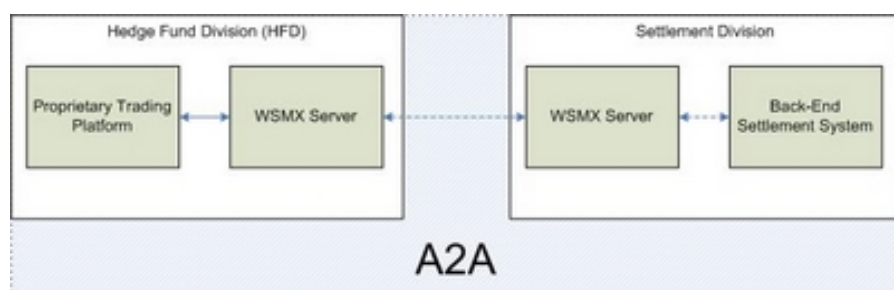


Figure 4: A2A architecture (click to enlarge)

Architecture: (see Figure 5)

To exploit the full potential of WSMX in A2A scenarios the company has two architecture options. Firstly it can implement WSMX on top of its system architecture in a global scale and introduce a policy that every division has to expose every Web Service capability in this central repository. Furthermore the adapters to back-end applications are built by the global IT department. Because this approach to introduce integration platforms on a strategic level requires a tightly cooperation between several IT departments and divisions in general, it proved to be not successful in the past in several cases in EAI system implementations. Since WSMX can act in a distributed manner, we assume for our use case that every single division operates its own WSMX.

As the term A2A implies, two applications have to communicate with each other. Additionally to these two applications, in our case a proprietary trading platform in the hedge fund division and a settlement system installed in the homonymous division, as mentioned WSMX runs in both divisions. Hence both divisions use WSMX as their mean to expose functionality either over the intranet or the extranet (internet). This assumption is solely made to utilize the repository of each WSMX. From the architectural point of view the use case would be nearly similar if only the HFD has WSMX installed and the settlement division only exposes the functionality of their back-end application as a Web Service. To be traceable this Web Service must be registered in at least one WSMX within the company. Since it is more likely that the division would do this registration in software installed in their own system architecture and as WSMX provides a central point of integration if they have to connect the settlement system to other systems within the division, we assume that WSMX is installed in the HFD as well as in the settlement division.

4.1.5. How WSMX can be utilized

Let's assume that the above mentioned HFD (hedge fund division) wants to buy one million shares of the Bridgestone Corporation, listed on the NIKKEI index and traded at the Tokyo Stock Exchange (TSE).

1. The proprietary application managing the portfolio of the HFD was configured in a way that it automatically has to order one million shares of the Bridgestone Corporation if they share price drops below 10 Yen. Since the application is configured to avoid tight coupling with specific applications, it sends the goal to buy one million Bridgestone shares at the current market price at the TSE to the *WSMX Adapter*.
2. The message is picked up by the *Communication Manager* and translated into an internal representation using the Compiler.
3. The Matchmaker first tries to find a capability stored in its own WSMX repository. Thus this is the first transaction in this stock the WSMX repository will most likely not include a stored capability description. Hence WSMX will try to match with capabilities stored in other WSMX. In contrast to the B2B use case this communication will be restricted to WSMX servers running within the entity bank. This restriction is not only set to tailor this use case to an A2A one, but in fact because the bank would strongly prefer to handle the order in a division belonging to itself. Only if the *Matchmaker* could not find any Web Service capability within the bank which offers to buy this share on the specified market WSMX would enlarge to outside capability repositories. The further steps in this case are the same as described above.
4. WSMX can similar to the B2B use case either find none, one or many Web Services. Now either the result set of service descriptions matching the goal are returned to the *WSMX Adapter* of the trading platform or the selection process of WSMX selects a service based on non-functional properties and preferences configured in WSMX at design time. Mostly even in an intra company interaction the price of the Web Service will be the most important factor to determine the selection process, because although the money costs for execution remain within the company, it is added to internal cost accounting of the HFD.
5. The *Communication Manager* generates a WSML message containing the capability descriptions of each found Web Service and sends it back to the *Adapter* of the WSMX server running at the HFD.
6. The selection is done in the Trading Platform by predefined preferences or a user interaction has to take place.
7. The remaining steps, again conform to step number 7 and 8 in the B2C use case. (see [Section 3.1.5](#))

4.1.6. Advantages WSMX exposes for the A2A use case

- As it is common in large-scale companies that divisions are not aware of all the functionalities other divisions offer computer process able within applications, the first main advantage of WSMX is to find these capabilities as long as they are stored in a WSMX repository. In the same process WSMX can execute them and helps to reduce the time for completing the full process fundamentally by avoiding any user interaction.
- Even if divisions would know where to find a specific functionality (it is quite likely that the HFD would know which settlement division is capable to process this trade), they don't have to know which specific application in the settlement division has to be addressed to invoke the trade. They only have to provide the goal and employ WSMX to find the address of the required application.
- Since WSMX don't rely on global ontologies like this is often the case with global EAI platforms, the several division can use their own ontologies in expressing the capabilities

of their Web Services and nevertheless they are still traceable by other divisions using different ontologies describing their goals.

- As mentioned in the B2B use case the settlement division is not bound to provide the functionality by one specific Web Service. If they change the back-end settlement system, the Web Service will still be found, as long as the *Capability Repository* is updated. Again the HFD still would have to change the configuration of their Trading Platform, since the instance data which has to be provided may change, but due to the formal specification this is possible without the settlement divisions assistance..

5. Conclusions and Further Directions

In this use cases we described the execution flow of different invocation scenarios and we have shown how different entities can benefit from using WSMX as their mean for executing Semantic Web Services. WSMX makes optimal use of the semantic descriptions offered by WSMO to address the requirements from the B2C, B2B and A2A domain. WSMX can be used as the centerpiece of a robust and flexible Web Service architecture: customers can use the client side editor to search and invoke web services, service requesters can use it to achieve their business tasks and it can be used to interconnect applications.

In the first release of WSMX not every component can already perform all the tasks described in the use cases. So far we have not implemented a client side editor for designing and sending goals. The functionality of the Choreography Engine is based on a simple match without any Mediation Service if the communication pattern of the requester and provider are different. Hence communication can only be performed if the two communication patterns are perfectly symmetrical.

The composition of Web Services was omitted in these use cases so far. A composition component would be responsible for executing complex compositions of services in order to achieve a certain goal. Since this functionality is crucial for an applicability in B2B scenarios we are currently investigating different language for specifying these compositions in WSMX as well as in WSMO.

Another issue we haven't implemented so far is the selection of a specific Web Service out of the result set in the matchmaking phase. Currently the selection has to be performed by the respective user or the requesting application. A selection within WSMX would have to consider non-functional properties, preferences, Service Level Agreements etc.

Further developments have also to be done to provide an extensive adapter framework. This especially includes an adapter development kit to allow WSMX implementers to create adapters for their own proprietary back-end applications.

References

[Bussler, 2003] C. Bussler: B2B Integration, Concepts and Architecture. *Springer-Verlag*, 2003, ISBN 3-540-43487-9.

[Keller et al., 2004] U. Keller, R. Lara, A. Polleres, H. Lausen, M. Stollberg, M. Kifer: Inferencing Support for Semantic Web Services: Proof Obligations. *WSML Working Draft v0.1*, 2004. Available from <http://www.wsmo.org/2004/d5/d5.1/v0.1/>.

[Linthicum, 1999] D. S. Linthicum: Enterprise Application Integration. *Addison-Wesley*, 1999, ISBN 0-201-61583-5.

[Moran/Zaremba, 2004] M. Moran, M. Zaremba: WSMX Architecture. *WSMX Working Draft v0.1*, 2004. Available from <http://www.wsmo.org/2004/d13/d13.4/v0.1/20040622/>.

[Oren, 2004] E. Oren: WSMX Execution Semantics. *WSMX Working Draft v01*, 2004. Available from <http://www.wsmo.org/2004/d13/d13.2/v0.1/>.

[Oren et al., 2004] E. Oren, M. Zaremba, M. Moran: Overview and Scope of WSMX. *WSMX Working Draft v0.1*, 2004. Available from <http://www.wsmo.org/2004/d13/d13.0/v0.1/20040611/>.

[Oren et al., 2004b] E. Oren (Ed.): BNF grammar for WSML language. *Working Draft v0.2*, 2004. Available from: <http://www.wsmo.org/2004/d16/d16.1/v0.2/>.

[Roman et al., 2004] D. Roman, H. Lausen, and U. Keller: Web Service Modeling Ontology Standard. *WSMO Working Draft v02*, 2004. Available from <http://www.wsmo.org/2004/d2/v02/20040306/>.

[Ruh et al., 2000] W. Ruh, F. X. Maginnis, W. J. Brown: Enterprise Application Integration: A Wiley Tech Brief. *John Wiley and Sons*, 2000, ISBN 0-471-37641-8.

[Stonebraker, 2002] M. Stonebraker: Too Much Middleware. *ACM SIGMOD Record* 31(1): 97-106, 2002.

Acknowledgement

The work is funded by the European Commission under the projects [DIP](#), [Knowledge Web](#), [Ontoweb](#), [SEKT](#), [SWWS](#), [Esperonto](#), and [h-TechSight](#); by [Science Foundation Ireland](#) under the [DERI-Lion](#) project; and by the Vienna city government under the [CoOperate](#) program.

The editors would like to thank to all the [members of the WSMX working group](#) for their advice and input into this document.



[webmaster](#)