Announcement
At the moment, the major focus of this deliverable is the elaboration of a general Web Service Discovery mechanism within WSMO. The functionality of such a mechanism is highly interrelated with the modeling decisions for WSMO Goals and WSMO Web Service Capabilities, as the modeling of these constructs has to support the discovery mechanism in the end. Currently, different possibilities are under discussion.

For development and testing, we stick to the VTA use case described in section 3.1 of this deliverable. The proposals elaborated in the working group can be accessed via the WSMO CVS web-interface at: http://cvs.deri.at/cgi-bin/viewcvs.cgi/wsmo//d3/d32/resources/, further explanations are given below. Everybody is invited to follow this process, feedback or active participation is always welcome.

When there is a final agreement on these issues, this deliverable will be completed and proposed for a final work draft of version 0.1.

The editors

D3.2 v0.1. WSMO Use Case Modeling and Testing

WSMO Working Draft 30 April 2004

This version:
http://www.wsmo.org/2004/d3/d3.2/v0.1/20040430/

Latest version:
http://www.wsmo.org/2004/d3/d3.2/v0.1/

Previous version:
http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/

Editors:
Michael Stollberg
Sinuhé Arroyo

Authors:
Michael Stollberg
Sinuhé Arroyo
Abstract

This document exemplifies the usage of the Web Service Modeling Ontology WSMO for modeling possible Web Service driven applications. The intent of this document is to exemplify use cases with usage scenarios of Semantic Web Services on the one hand, and on the other to showcase modeling with WSMO as an evaluation with real-world testing as support for recursive development of WSMO. For use case modeling, we stick to the latest final working draft of Web Service Modeling Ontology WSMO, Version 0.2.

Related Documents

WSMO Standard: D2 v0.2 Web Service Modeling Ontology (WSMO) , current version at: http://www.wsmo.org/2004/d2/v0.3/

WSMO Primer: D3.1 v01. WSMO Primer


Table of contents

1. Introduction
   1.1 Semantic Web Services
   1.2 The Web Service Modeling Ontology WSMO

2. Use Cases
   2.1 B2C - Virtual Travel Agency
      2.1.1 Description
      2.1.2 Scope
      2.1.3 Actors, Roles and Goals
      2.1.5 System Architecture
   2.2 B2B - Integration with Semantic Web Services
1. Introduction

This document exemplifies the usage of the Web Service Modeling Ontology WSMO for describing relevant aspects for Semantic Web Services. Therefore, we describe possible use cases of Semantic Web Services and showcase how these can be modeled with WSMO, especially for support of the Semantic Web Service usage scenarios in particular use cases. We briefly replicate the objectives and the approach of WSMO and outline use cases within possible usage scenarios of Semantic Web Services. Then, we showcase how specific use cases can be modeled in WSMO along with explanations on the modeling decisions. Besides, we provide the WSMO models in a computational format.

This Deliverable is intended to evolve in accordance to the ongoing development of the WSMO project, serving as a testing environment and providing input for a recursive, real world testing development of WSMO. In the longer run, additional use cases will be added in order to widen possible solutions for Semantic Web Service technologies around WSMO.
This document is organized as follows: the remainder of Section 1 replicates the objectives and approach of WSMO; Section 2 discusses possible application areas of Semantic Web Services. Section 3 provides the modeling of the use cases in WSMO, pointing out the WSMO approach for Semantic Web Service technologies. Section 4 concludes the document. The complete WSMO models as computational resources are provided in the Appendices.

The use case modeling in this document relies on the latest final working draft of the Web Service Modeling Ontology WSMO, Version 0.2.

### 1.1. Semantic Web Services

A Web Service is a piece of software accessible via the Web. Current Web Service technologies allow exchange of messages between Web Services [SOAP], describing the technical interface [WSDL], and advertising a Web Services in a registry [UDDI]. These technologies do not provide any information about the meaning of information used, neither do they explicitly describe the functionality of a services as needed for automated usage of Web Services. There are also an alternative and complement specifications such as for example [ebXML], which are still perceived to have much potential for widespread adoption as Web Services.

ebXML has become the global electronic business specification, that defines a framework for global electronic business. Enhanced Web Service technologies aim at more sophisticated techniques to describe Web Services, emphasizing the concept of Semantic Web Services. In our understanding, a Semantic Web Service is defined as a “self-contained, self-describing, semantically marked-up software resources that can be published, discovered, composed and executed across the Web in a task driven automatic way” [Arroyo et al., 2004]. By machine-processable descriptions of the relevant information and by means of automated mechanisms that utilize this information, the following functionalities for Web Services shall be achieved.

- **Automatic Web Service Discovery**: finding Web Services that abide to a service requester's specification of a desired functionality
- **Automatic Web Service Composition**: assembly of services based on its functional specifications in order to achieve a given task and provide a higher order of functionality
- **Automatic Web Service Execution**: invocation of a concrete set of services, arranged in a particular way following programmatic conventions that realizes a given task.

### 1.2 The Web Service Modeling Ontology WSMO

The aim of WSMO and its surrounding efforts is to define a coherent technology for Semantic Web Services (short: SWS). WSMO defines the modeling elements for describing several aspects of Semantic Web Services. The conceptual basis of WSMO is the Web Service Modeling Framework [WSMF], wherein four main components are defined that are needed for a full coverage framework for Semantic Web Services (see Figure 1). The first component is Ontologies which provide the formal semantics of the information used by all other components. The second component is Goals that specify objectives that a client may have when he consults a web service. The third component is Web Services. For supporting automated
discovery, composition, and execution of Web Services, descriptions are required on the functionality provided by a Web Service (called “Capability” in WSMO). For supporting automated choreography and execution compensation of Web Services, particular information on the external visible behavior of a Web Service are needed (called “Interface” in WSMO), including information on the technical accessibility and the actual message exchange of Services. The fourth component of WSMO is Mediators, which are used as connectors between particular components and include possibly required mediation facilities needed to make connected components interoperable. WSMO distinguishes different types of Mediators. The components of WSMO along with exhaustive explanations are presented in the WSMO Primer.

![Figure 1. WSMO Components](image)

## 2. Use Cases

Semantic Web Services can be used in manifold application fields. In accordance with the use cases defined in [Web Services Architecture Usage Scenarios](https://www.w3.org/2002/ws-uc/usage-scenarios.html) by the [W3C Web Services Architecture Working Group](https://www.w3.org/2002/ws-aq/), we discuss two most common use case scenarios to exemplify the usage of SWS technologies:

1. A "Virtual Traveling Agency" that provides end-user services for e-Tourism by aggregating Web Services of different tourism service providers. This is a “B2C” use case, i.e. wherein a third party provides a service to end users as a Client / Service model as an aggregation of Semantic Web Services.
2. The second example is concerned with B2B Integration wherein a business entity, e.g. a business document, is exchanged between enterprises. Therein, different aspects of EAI might arise which shall be handled by Semantic Web Services technology.

For describing the use cases, we slightly modify the methodology of the W3C Use Case descriptions and extend by the requirements arising for Semantic Web Services technologies. The following lists the aspects we use for the use case definitions below.

- **Description**: describes the overall scenario
- **Scope**: defines the scope of the application scenario described
• **Actors, Roles and Goals:** identifies the actors in the scenario, their roles (i.e. what they do in the scenario) and their goals (i.e. what they want to achieve by participating in the scenario).

• **Usage Scenarios:** the W3C Service Architecture Working Group defines a [use case](#) as "... a sequence of interactions between a service requestor and one or more services, which achieve measurable results for the requestor", and a [usage scenario](#) as "... an atomic step in a path through a use case", i.e. an activity that has to performed during execution of the use case and which can be automated by appropriate Semantic Web Service technologies. For each use case we describe the particular usage scenario by the following information:
  - participating actors and their goals
  - activities to be performed
  - technological requirements for this
  - and possible extensions of the scenario.

• **System Architecture:** In addition to the use-case oriented aspects of the W3C methodology, we also outline the general requirements and possible architecture of the respective SWS-based application.

### 2.1 B2C - Virtual Travel Agency

In [Web Services Architecture Usage Scenarios](#), the travel agency use case is separated into two use cases - one with static discovery and one with automated discovery. With Semantic Web Services we clearly want to support automated discovery, thus we restrict the first WSMO use case to a Virtual Travel Agency scenario that supports automated discovery of Web Services.

#### 2.1.1 Description

Imagine a “Virtual Traveling Agency”, called VTA for short, that is an end user platform providing eTourism services to customers. These services can cover all kind of information services concerned with tourism information - from information about events and sights in an area to services that support booking of flights, hotels, rental cars, etc. online. Such VTAs are already existent, but as this point of time they mostly are an information portal along with some web-based customer services (e.g. [eTourism.at](#)). By applying Semantic Web Services, a VTA will invoke Web Services provided by several eTourism suppliers and aggregate them into new customer services. Such VTAs will provide automated eTourism services to end users, thus tremendously enhancing the functionality of currently existing VTAs.

The overview of the use case for VTAs that aggregate Web Services of different tourism service providers looks like this: a customer uses the VTA as the entry point for his request. This request must fit to an end-user service that the VTA provides. These end-user services are aggregated by the VTA by invoking and combining Web Services offered by several tourism service providers. Therefore, there must be some kind of contract between the service providers and the VTA for regulating usage and allowance of the Web Services. Figure 2 shows this overview (modified and extended from [W3C Travel Agent Use Case overview](#)).
2.1.2 Scope

The overview described above can be seen as a general structure for VTAs that can be extended to more complex scenarios wherein the customer can be a Web Service itself, thus creating a network of composed services that offer complex tourism services. For example, one VTA can provide flight booking services for an airline union, another VTA aggregates booking service for a worldwide hotel chain, and a third VTA provides booking services for rental cars by combining the services of several worldwide operating car rental agencies. Then, another VTA uses these VTA-services for providing an end-user service for booking complete holiday trips worldwide.

In order to showcase and test the applicability of WSMO and not to get lost in real-world modeling of eTourism use cases, we restrict ourselves to a simple VTA use case from booking international online train tickets. This use case is described in more detail in section 3.1.VTA for International Online Train Ticket.

2.1.3 Actors, Roles and Goals

In general use case there are 3 actors. The following defines what they are, why they participate in this use case (goal), and with whom they need to interact in what way (role).

1. **Customer**: the end-user that requests an end-user service provided by the VTA
   - **Goal**: automated resolution of the request by a user-friendly tourism service
   - **Role**: end-user, interacts with VTA for service usage, payment, and non-computational assets (e.g. receiving the actual ticket when booking a trip)

2. **Tourism Service Provider**: a commercial company that provides specific tourism services
   - **Goal**: sell service to end customers, maximize profit as a commercial company
- **Role**: provides tourism service as a Web Service (also provides the necessary semantic descriptions of the Web Services), has a usage and allowance contract with the VTA

3. **VTA**: the intermediate between the Customer and the Tourism Service Provider. Provides high-quality tourism services to customers by aggregating the separate services provided by the Service Providers.
   - **Goal**: provide high-quality end-user tourism services, use existing tourism services and aggregate them into new services, maximize profit as a commercial company / represent union of service providers (depending on the owners of the VTA).
   - **Role**: interacting with customer via user interface (can be web-based for human customers or and Interface / API for machine-users), usage and allowance contract for Web Services offered by Service Providers, centrally holding all functionalities for handling Semantic Web Services (mechanisms for discovery, composition, execution, etc.)

### 2.1.4 Usage Scenarios

We identify the following usage scenarios

1. **VTA interacts with Service Providers on contract and Web Service usage and allowance**
   - **Participating Actors**: VTA and Service Providers
   - **Activities**: business contract negotiation
   - **Technological Requirements**: contract information is displayed in system, i.e. Web Service usage is implemented via Policies
   - **Possible Extensions**: contract negotiation can be supported by automated mechanisms

2. **Customer requests VTA for searching tourism service offers, VTA detects suitable Web Services for searching tourism service offers and displays results to Customer**
   - **Participating Actors**: Customer and VTA
   - **Activities**:
     1. Customer selects "Search" services as provided by the VTA
     2. VTA discovers, invokes and executes corresponding Web Services
   - **Technological Requirements**:
     1. VTA has to pre-define a "Search" functionality that can be requested by a Customer
     2. Web Services must be semantically described in order to support dynamic discovery (assuming that single Web Services can perform the search functionality)
     3. VTA has to hold mechanisms for automated Service Discovery
   - **Possible Extensions**:
     - the Customer specifies his request in natural language and the requested VTA-service is detected automatically
     - several Web Services are aggregated for functionality

3. **Customer selects a concrete offer and requests booking for this offer (interacting with the VTA), VTA detects and aggregates Web Services for booking (incl. booking, payment, etc.), displays result to Customer and handles complete execution of customer-interaction (computational part)**
   - **Participating Actors**: Customer and VTA
- **Activities:**
  1. Customer selects one concrete offer out of the Search results of usage scenario 2
  2. VTA discovers and composes available Web Services from Service Providers for
  3. VTA executes the Web Services in the sequence determined, controls the execution (handles errors and detects alternative paths if a Web Service fails)
  4. VTA interacts with Customer during execution when further information is needed (e.g. a creditcard number for payment)

- **Technological Requirements:** contract information is displayed in system, i.e. Web Service usage is implemented via Policies
  1. Web Services must be semantically described in order to support dynamic discovery, composition, and execution
  2. VTA has to hold mechanisms for automated Service Discovery, Composition, and Execution
  3. VTA has to provide and interaction interface for contingent Customer-interaction during Service execution

- **Possible Extensions:** advanced mechanisms for automated execution of aggregated Web Services

4. **VTA interacts with Customer and Service Provider for non-computational parts (e.g. delivery of actual tickets)**

- **Participating Actors:** Customer, VTA

- **Activities:** customer notification, accounting, good delivery (out of computational system), etc.

- **Technological Requirements:** mechanisms for notification and accounting

- **Possible Extensions:** Web Services can be used for:
  o customer notification
  o VTA-Service Provider interaction on accounting and good delivery mandate

### 2.1.5 System Architecture

In this use case, the VTA is the central point of interaction between the Customer and Web Services. Regarding the technological requirements, it gets obvious from the Usage Scenario descriptions that (1) the Web Services offered by the Service Providers have to carry sufficient descriptive information to support automated Web Service usage, and (2) that the VTA has to hold all mechanisms to handle Semantic Web Services. The basic architecture of such a VTA as a central entity for Semantic Web Services handling is shown in Figure 3. The essential functionalities of Semantic Web Service enabled VTAs – with special regard to the requirements for Semantic Web Service technologies – are:

- It has to provide a user interface for customer interaction (for both human and machine users)
- It has to hold generic end-user services that users can “instantiate”
- It has to discover suitable Web Services for an “instantiated” user request
- It has to invoke and combine external Semantic Web Services
- It has to provide a Web Service Execution Environment with control functions, error handling, and support for optional user interaction
• It has to have to deal with properly heterogeneous resources, thus holding appropriate mediation facilities.
• It has to provide an Interface for cooperation with Service Providers.

Figure 3. General Architecture of a SWS-enabled VTA

Summarizing, the VTA is a SWS-enabled B2C application that provides an end-user service following a C/S Model. In order to support coherent functionality of the VTA and ensure that the descriptions of Web Services are compatible to this, an overall framework for SWS technologies is needed. This is provided by WSMO. Section 3.1 exemplifies the modeling of the WSMO components for a real world VTA use case in detail.

2.2 B2B - Integration with Semantic Web Services

The second use case is concerned with integration of possible heterogeneous resources in B2B settings which is considered as one of the most important application fields of Web Service technology.

2.2.1 Description
In the B2B use case, two enterprises called E1 and E2 want electronically exchange business documents across the network. It is assumed that partners may not know each other before carrying business transaction and that is why contract negotiation and contract agreement are essential elements of this use case. It is assumed that each of the partners expose a set of web services with the given capabilities, which can handle conversation using any possible known B2B protocols (e.g. ebXML, RosettaNet etc.). The contract agreement defines roles of enterprises in the conversation e.g. one of the enterprise E1 becomes the seller and the second enterprise E2 becomes the buyer. Agreement also predefines the order of the message exchange pattern e.g. buyer first sends purchase order (PO) and after that it receives purchase order acknowledgement (POA). Differently than in the previous B2C use case, where the client/server model of interactions has been adopted, the peer-to-peer model is used in this use case - partners are equal and they carry the conversation. Each of the companies has own orchestration and the set of web services, which enables to exchange business documents electronically. Infrastructure provided by SWS takes care for any necessary mediation between web services (links web services), ontologies (resolves possible representation mismatches between ontologies used by these two enterprises), goals (links goals) and web services and goals. SWS infrastructure supports the execution of the contract to fulfill approved agreement.

![Figure 4. B2B Integration with Semantic Web Services](image)

In this use case an ultimate goal of an enterprise E1 is to integrate its own back-end system with the back-end system of an enterprise E2. Once integrated, SWS software enables back-end systems of both companies to interact and to preserve the message, process and protocol semantic. The information systems used by enterprises E1 and E2 are **autonomous**, **heterogenous** and **distributed**. Semantic Web Services address each of these three properties and the software based on SWS enables companies to cooperate.
The back-end systems in E1 and E2 are *autonomous* since each of them changes its state without informing other system about it. SMS software enables to track state changes of back-end applications to facilitate coordination between systems of E1 and E2.

The back-end systems in E1 and E2 are *heterogeneous*, because each of them has different conceptual model for expressing business semantics. SWS software takes care of appropriate mediation of the representation and meaning of the back-end system to the equivalent representation and meaning of the other system. The SWS software ensures to maintain the same semantics between back-end systems of E1 and E2.

The back-end systems in E1 and E2 are *distributed* because each of them maintains its own state independently from the other system. Back-end applications in companies E1 and E2 do not share data or state at all. SWS software implemented in both companies takes care of transporting data between the systems.

### 2.2.2 Scope

The use case assumes peer-to-peer relationships between two business partners carrying conversation about purchasing/selling of goods. The B2B use case focuses on the technical infrastructure based on the SWS technology, which enable any business company to automatically discover web services which are capable to fulfill its goals, compose simple web services into complex web services to achieve a given goal and to automatically execute given services in a particular order. This use case assumes that there may be no prior business relationships between two enterprises before the discovery. Enterprise E1 must find enterprise E2 and they must agree and enforce the contract in their companies. Agreement should define roles of each of them in the agreed business process – e.g. one of them would become a buyer and one of them would become a seller. The agreement can lead to only one time execution of the agreed business process (e.g. request purchase order) or to long time relationships based on the multiply execution of the agreed contract. Payments are sent through financial institutions and at this stage they are out of the scope of this use case. The same situation concerns the shipment of the goods. This use case consider sending documents as for example purchase orders or invoices, but the physical shipment of goods is out of the scope of this use case.

### 2.2.3 Actors, Roles and Goals

There are two actors in the B2B use case – actors represent business entities. The size and the importance of companies are not predefined in this use case. They might differ in size but from the perspective of this use case it should not matter which one of them is a more dominant partner. Both of the enterprises undertake a predefined role in the use case. These are:

1. **Buyer**: the company, which initiates the use case by searching for a partner, which is capable to sell goods.
   - **Goal**: Finding a business partner, who is capable to provide goods. Signing the contract, discovering capabilities of the seller, composing provided web services and executing them.
   - **Role**: A business entity, which seeks any partner to achieve given goals in
business relationships. Once the contract is signed it must be executed and as
the result of contract execution, the buyer should receive goods. Buyer
initiates the process described in this use case.
2. **Seller** - seller provides goods. It waits for buyers, responds to their requests,
signs the contract and ships goods.
   - **Goal**: Providing goods. Signing the contract, discovering capabilities of the
     buyer, composing provided web services and executing them.
   - **Role**: A business entity, which waits for the partner to establish business
     relationships. As the result of the execution of the contract, the seller should
     send goods the seller.

### 2.2.4 Usage Scenarios

In this use case the following usage scenarios have been identified:

1. **Contract negotiation and implementation of agreement between buyer and
   seller.**
   - **Participating actors** - buyer and seller
   - **Activities** - business contract negotiation and implementation
   - **Technological Requirements** - The technology should enable matching
     goals of a buyer with capabilities of a seller. But matching goals of capabilities
     is not sufficient, because once goal is matched with the capability, the
     interfaces of two businesses should be matched as well.
   - **Possible Extensions** - contract is negotiated and implemented completely
     automatically by appropriate infrastructure

2. **Typical business messages exchange (e.g. PO & POA exchange);**
   - **Participating actors** - buyer and seller
   - **Activities** - buyer sends PO to seller. Buyer can at any time check the status
     of processed order. Seller sends back POA.
   - **Technological Requirements** -
   - **Possible Extensions** -

3. **SWS infrastructure crashes - once it recovers, it reliable commence its
   operations**
   - **Participating actors** - buyer and seller
   - **Activities** -
   - **Technological Requirements** -
   - **Possible Extensions** -

4. **E1 and E2 want to deploy a new integration type. The developer responsible
   for the SWS software writes a new integration type, which is next deployed by
   SWS infrastructures in both enterprises.**
   - **Participating actors** - buyer and seller
   - **Activities** -
   - **Technological Requirements** -
   - **Possible Extensions** -

### 2.2.5 System Architecture

Web Services Modeling Execution (WSMX) is the infrastructure hosted by each of
the enterprises to support services following a peer-to-peer model. WSMX is software
implementation of a web service execution environment supporting the development,
management and execution of Semantic Web enabled Web Services. WSMX
platform does not differentiate between calls coming from the back-end application systems (intra-company information systems) and from the information systems of other enterprises. WSMX can also communicate directly with other WSMX platforms hosted by other enterprises as shown on figure 5.

![Figure 5. B2B Use Case System Architecture](image)

### 3. WSMO Use Case Modeling

In the following we model the two use cases described above in WSMO in order to explicate the practical usage and the design of WSMO. We apply the following methodology for this:

1. describing the use case, especially the functionality of the Semantic Web Services handling component of the system
2. define a list of requirements on what has to be known to the application in order to make SWS applicable
3. model the use case in WSMO, including all building blocks of WSMO, along with detailed explanations of the modeling and the building blocks of WSMO
4. all WSMO models as downloadable computational resources for testing of SWS-technologies developed on WSMO.

For step number 3, the models of the respective WSMO components are described in Listings in this document. Therein, the Listings specify the different WSMO components as conceptual models in accordance to the specifications in [WSMO Standard, V0.2](#). These models have to be transformed for usage within a specific technology, with respect to syntax and technological constructs. For step 4, we provide the models as computational resources for download, testing and development in the [Appendices](#). Currently, we support FLORA-2 as a F-Logic reasoner (see [FLORA-2 homepage](#)). The models provided for download in the Appendices are runnable as separate FLORA-2-programms; the structure and the connections of these programs as WSMO-components have to be defined by the application developer (e.g. transforming the "usedMediators"-constructs as well as...
namespaces in the respective technology support by the tool). Besides, the identifier of every WSMO element should be a URI. In the use case, the URL "http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/FILE" is the general ID for all models of the use case, wherein FILE is replaced by the actual file name of the component as linked to in the Appendices. The elements of specific components have then the identifier "../resources/FILE#element", which make the identifier-concept of WSMO conformant to the concept of URIs in web technologies. With regard to readability of the listings, we only specify the file name as the component identifier in the Listings.

For modeling of the WSMO components in the Listings in this document we use a human readable syntax, based on the syntax for F-Logic as defined in [Kifer et al., 1995]. Note that the Listings are not parsable in FLORA-2, they are only intended to illustrate the modeling. The parsable resources are provided in the Appendices. The axioms are provided in the executable FLORA-2 syntax; therein the following syntactical conventions hold:

- **Constants** are written in lower case letters
- **Variables** are written in upper case letters
- [ ] denotes property collections of concepts
- ( ) encloses arguments for methods and for predicates
- Arrows:
  1. => and =>> are used in attribute type definitions in class signatures (single and set valued). For simplicity we assume all attribute type definitions are inherited. In the original F-Logic syntax inheritance is expressed by "*=>, *=>>", but this behavior can be easily emulated by adding a rule: X[Y *=> Z] :- X[Y => Z] (analogous for "=>>").
  2. -> and --> are used in value definitions in F-molecules (single and set valued)
- : denotes class membership
- :: denotes subclass relationship
- , denotes a conjunction in an axiom, i.e. in F-Logic molecules
- ; denotes a disjunction in an axiom, i.e. in F-Logic molecules
- { } denotes a set of values
- , is a separator of values in sets, as well as a syntactical separator in signatures
- // denotes a comment

### 3.1 VTA for International Online Train Tickets

According to the general VTA use case described in Section 2.1 B2C - Virtual Travel Agency we define the following use case here:

- A customer wants to book an online ticket for an international train connection, more precisely from Innsbruck in Austria to Frankfurt in Germany.
- A VTA provides a search service for international train connections in Europe (here: only Austria and Germany) and an aggregated service for booking the tickets for international train connections in Europe online. (At design time of this use case the national train operators of Austria and Germany only provided online ticket booking facilities for national train connections; so we
assume that the VTA has to compose the online ticket booking services from the Austrian and the German train operators.)

- Both the Austrian national train operator “ÖBB” as well as the German national train operator “DB” provides Web Services for searching international train connections and for booking tickets for national train connections online. (These services exists as conventional Internet Services, see Figure 6; here we assume that they are provided as Web Services).

The course of the use case is the following:
- the customer poses a request for an international train connection from Innsbruck to Frankfurt on 23rd May 2004, at 16.00 local time
- the VTA returns a set of possible connections
- the user selects one of these connections and poses a request for booking the ticket online
- the VTA combines the online train ticket booking services from ÖBB and DB, executes the booking and payment process, and sends the online ticket per email to the Customer.

For the aggregated service, the VTA has to determine the itineraries of the international connections, and to split them at the border stations into national itineraries. The VTA has to mediate between the following web services:

- The timetable service, providing timetable information and itineraries on national and international connections.
- The national ticketing services, providing online tickets for national itineraries.

The rationale for choosing this use case is that it showcases a possible VTA use case as described above within all the components identified in WSMO. The components are simple, thus this use case allows showing the modeling of WSMO elements without getting lost in complicated definitions of specific elements.
3.1.1 Functional Requirements

The following lists the requirements analysis for modeling the use case. For each of the components of WSMO, a list of requirements is defined that are needed in order to enable the requested functionality of the VTA for online search and booking of international train tickets, regarding the use case described above.
**Table 1. Requirements Domain Ontologies**

| O1 | We need ontological information on international train itineraries and on the purchasing process. This information should be kept in separated ontologies, following the design principle of modular ontology design. |
| O2 | An itinerary is described by a Start- and End-Location, date and time of departure and arrival, the station where the border is crossed, and the fare. |
| O3 | There has to be customer that buys a train ticket |
| O4 | An itinerary describes a valid international train connection. |
| O5 | There exists a concept that defines whether a location is located at the border between 2 countries |
| O6 | A ticket is valid for exactly 1 itinerary |
| O7 | A ticket is valid for exactly 1 customer |
| O8 | A location description consists of a Location identifier, a Country identifier, and an indicator that the location has a train station |
| O9 | The purchase ontology has to identify the buyer and seller roles, a product with a price, and valid payment methods |
| O10 | The only valid payment method for online tickets is credit card payment |
| O11 | Information on Date and Time should be defined generally in a separate ontology |

**Table 2. Requirements Goals**

| G1 | Booking a Online Train Ticket |
| G1.1 | From Innsbruck to Frankfurt |
| G1.2 | Start time: 23rd May 2004, at 16.00 local time |

**Table 3. Requirements Web Services**

| W1 | Each National Train Operator provides a Web Service that offers an international train connection timetable and national online ticket booking |
| W2.1 | The international train connection timetable takes a start location and an end location and a departure date and returns a set of itineraries |
| W2.1.1 | Exceptions are: |
| | - Service not available |
| | - Start or End Location does not exists |
| W2.2 | The national online ticketing service takes an itinerary with start location and end location in the national country, a credit card number and returns a ticket for this itinerary. |
| W2.2.1 | Exceptions are: |
- Service not available
- Credit card not accepted

<table>
<thead>
<tr>
<th>Table 4. Requirements Mediators</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
</tr>
<tr>
<td>M2</td>
</tr>
<tr>
<td>M2.2</td>
</tr>
<tr>
<td>M3</td>
</tr>
</tbody>
</table>

### 3.1.2 WSMO Modeling

The following provides the modeling of the use case in WSMO with respect to the requirements determined above. The models are presented in the same structure as in the requirements analysis. As explained above, the Listings below specify conceptual models of the distinct WSMO components which have to be transformed into the technology provided by the tool used for reasoning on the ontologies. The full WSMO models as downloadable computational resources for this use case are provided in Appendix A. The models apply the syntax defined in section 3.

The use case modeling in this document relies on the Web Service Modeling Ontology WSMO, Version 0.3. This version of WSMO does not provide elaborated description elements for all WSMO components, especially for Web Services a sophisticated specification of the WSMO modeling primitives only exists for Web Service Capabilities. As we can only apply such elaborated description structures for use case modeling, we restrict this version of the use case modeling to the WSMO description elements defined.

#### Ontologies

We define 3 domain ontologies that provide the terminology definitions for the use case. The first ontology "International Train Ticket" describes the domain of train tickets, the second ontology "Date and Time" defines a general model for specifying time and dates and relationships of them, and the third ontology "Purchase" describes generic elements of purchasing a product between a buyer and a seller.

We apply the following conventions in the Listings for ontology specifications (those which are not ontology-specific hold for all other WSMO component specifications as well):

- **Identifier**: As defined above, WSMO applies URIs as identifiers for all WSMO elements, thus being conform to web technologies. All URIs for this use have the URL "http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/FILE" is the general ID for all models of the use case, wherein FILE is replaced by...
the actual file name (without file type extension) of the component as linked to in the Appendices. The elements of specific components have then the identifier "..../resources/FILE#element". For the ease for readability, we only specify the FILE-name in the use case Listings. Moreover, we only specify the identifier for the top-level WSMO elements explicitly, as the identifiers for the sub-elements are implicitly defined.

- "usedMediators"-element: this is the general construct for connecting different WSMO components. According to specifications in WSMO Standards, only specific types of Mediators can be WSMO "imported" into a specific WSMO component. In general, a WSMO Mediator handles all integration and mediation issues required in between the connected resources. Especially OO Mediators, which allow to use an ontology as the terminological basis in a WSMO component, resolve all ontology integration aspects (including namespaces, resolution of heterogeneities, etc.). The targetComponent of a OO Mediator, i.e. the user of an ontology, receives his required information space, meaning all knowledge that is provided within the OO Mediator, and simply uses this without regard to the actually used ontology. The ontologies below are connected as the use constructs of each other. Therefore OO Mediators are defined. These, along with further explanations on the used Mediators in the use case, are defined in Mediators below.

- Axiom Definitions: all axioms in the Listings are written in FLORA-2 syntax, see above for syntactical conventions. The axioms are the same as provided in the executable resources in Appendix A.

- Ontology Notions: In the ontology specifications below, we only specify the corresponding WSMO elements for non-functional properties, usedMediators, and axiomDefinitions. For the other ontology notions the semantics of F-Logic are exactly the same as the ontology modelling primitives defined in WSMO Standard, thus we do not have to specify the correspondence to the WSMO Standard meta-ontology explicitly.

The "International Train Ticket" Ontology defines an itinerary and the surrounding concepts as defined in Listing 1. Additionally, a axiom is defined that checks the validity of the traveling dates (the start date / time has to be in the future and the arrival date / time as to be later than the starting date / time) as well as some instances needed in the further use case modeling. Listing 1 shows the F-Logic specification of the ontology.

<table>
<thead>
<tr>
<th>Listing 1. Domain Ontology &quot;International Train Ticket&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>// International Train Connections Domain Ontology</td>
</tr>
<tr>
<td>// Non-functional Properties (Core Properties)</td>
</tr>
<tr>
<td>title -&gt; 'International Train Connections Domain Ontology'</td>
</tr>
<tr>
<td>creator -&gt; 'DERI'</td>
</tr>
<tr>
<td>subject -&gt; 'International, Train Itineraries, Ticket Booking'</td>
</tr>
<tr>
<td>description -&gt; 'International Train Itineraries for Online Ticket Booking'</td>
</tr>
<tr>
<td>publisher -&gt; 'VTA'</td>
</tr>
<tr>
<td>contributor -&gt; 'Michael Stollberg, Rubén Lara, Holger Lausen, Axel Polleres'</td>
</tr>
<tr>
<td>date -&gt; 20040419</td>
</tr>
</tbody>
</table>
usedMediators
// Use Mediators for using the Date and Time Ontology
VTA-OOM-trainConnection

// Concepts

country[
    name => string
].

location[
    name => string,
    country => country
].

station[
    locatedIn => location
].

// Concept borderStation defines a train station at the border
borderStation::station[
    border => country
].

itinerary[
    startLocation => location,
    endLocation => location,
    borderStation => borderStation,
    departure => dateAndTime, // from Date and Time Ontology
    arrival => dateAndTime // from Date and Time Ontology
].

customer[
    name => string
]
// Concept ticket defines a ticket that relates an itinerary to a customer
ticket[
    itinerary => itinerary,
    customer => customer
].

// Axioms

// departure has to be before arrival
I:itinerary[departure->X2, arrival->X3], X2[after(X3)].

// Instances (needed throughout the use case modeling)
geography:country[name -> 'Germany'].
austria:country[name -> 'Austria'].

innsbruck:location[name -> 'Innsbruck', country -> austria].
frankfurt:location[name -> 'Frankfurt', country -> germany].
kufstein:location[name -> 'Kufstein', country -> austria].
salzburg:location[name -> 'Salzburg', country -> austria].
kufsteinHbf:borderStation[locatedIn -> kufstein, border -> {austria, germany}].
salzburgHbf:borderStation[locatedIn -> salzburg, border -> {austria, germany}].

The "Date and Time" Ontology in Listing 2 defines models for dates (i.e. certain days) and time (i.e. definition of certain points in time). Further, it defines axioms that represent conventional aspects of date and time, like ‘before’ and ‘after’, etc. In the use case, this is needed to determine validity of train connections, e.g. for ensuring that a ticket is not for an itinerary that is in the past. It also can be used generally for expressing dates and time and relationships between them. Listing 2 only displays the ontology schema and the algebra for date and time, while the downloadable file contains instances and queries for testing.

The main ontology taken into consideration for developing this representation in F-Logic is an entry sub-ontology of time, available at http://www.isi.edu/~pan/damltime/time-entry.owl. This ontology uses abstract temporal concepts like instant, interval and event and uses the Gregorian calendar as representation (partly using own encoding and partly using XSD encoding). Axioms are defined in first order logic in the accompanying paper [Pan and Hobbs]; there also is a LISP version of these axioms available at http://www.cs.rochester.edu/~ferguson/daml/daml-time-20030728.lisp. Other ontologies like COBRA calendar clock ontology (http://daml.umbc.edu/ontologies/cobra/0.4/calendar-clock) are only a straight forward representation of the Gregorian calendar, without any abstraction of concepts and description of axioms. Widely used concrete representations for date and time are defined in ISO 8601 (Numeric representation of Dates and Time) and in the XML
Schema Definition ([http://www.w3.org/TR/xmlschema-2/](http://www.w3.org/TR/xmlschema-2/)), which is based on ISO 8601. This ontology uses the Gregorian calendar with the representation based on the definition in [http://www.w3.org/TR/xmlschema-2/](http://www.w3.org/TR/xmlschema-2/).

---

**Listing 2. Domain Ontology “Date and Time”**

```xml
// Date and Time Ontology

// Non-functional Properties (Core Properties)
title -> 'Date and Time Ontology'
creator -> 'DERI'
subject -> 'Date, Time, Date and Time Algebra'
description -> 'generic representation and algebra for date and time'
publisher -> 'VTA'
contributor -> ' Holger Lausen, Rubén Lara, Axel Polleres'
date -> 20040430
type -> 'domain ontology'
format -> text
language -> 'English'
relation -> http://www.isi.edu/~pan/damltime/time-entry.owl,
http://daml.umbc.edu/ontologies/cobra/0.4/calendarclock,
http://www.w3.org/TR/xmlschema-2/
coverage -> 'general'
rights -> 'DERI'
version -> 1.8

// Concepts and Axioms

// An instant represents a particular point in time [1]
instant[].

// An interval represents a duration between 2 points in time [1]
interval[].

// DATE

// the Class date and its representation according to the
Gregorian calendar
date:instant[
   dayOfMonth=>dayOfMonth,
   monthOfYear=>monthOfYear,
   year=>year
 ].

// Axiom for a valid date
X[invalid] :- X:date, (X.dayOfMonth[invalid];
X.monthOfYear[invalid]; X.year[invalid]).

//a dayOfMonth is represented by an integer
dayOfMonth::integer.
//integrity constraint for valid dayOfMonths:
X[invalid] :- X:dayOfMonth, (X<0; X>31).

//a year is represented by an integer
year::integer.
//integrity constraint for valid years: none (negative years
are allowed)

//a monthOfYear is represented by an integer
monthOfYear::integer.
//and has additional properties, concrete month are defined as
instances according to this class definition:
monthOfYear[
    name=>string,
    daysAfterBeginOfYear=>integer
].
//integrity constraint for valid monthOfYear:
X[invalid] :- X:monthOfYear, (X<0; X>12).

// Instances for the individuel months of a Year
1:monthOfYear[daysAfterBeginOfYear->31, name-> 'January'].
2:monthOfYear[daysAfterBeginOfYear->59, name-> 'February'].
3:monthOfYear[daysAfterBeginOfYear->90, name-> 'March'].
4:monthOfYear[daysAfterBeginOfYear->120, name-> 'April'].
5:monthOfYear[daysAfterBeginOfYear->151, name-> 'May'].
6:monthOfYear[daysAfterBeginOfYear->181, name-> 'June'].
7:monthOfYear[daysAfterBeginOfYear->212, name-> 'July'].
8:monthOfYear[daysAfterBeginOfYear->243, name-> 'August'].
9:monthOfYear[daysAfterBeginOfYear->273, name-> 'September'].
10:monthOfYear[daysAfterBeginOfYear->304, name-> 'October'].
11:monthOfYear[daysAfterBeginOfYear->334, name-> 'November'].
12:monthOfYear[daysAfterBeginOfYear->365, name-> 'December'].

// TIME

// Concept Time
time:instant[
    hourOfDay=>hourOfDay,
    minuteOfHour=>minuteOfHour,
    secondOfMinute=>secondOfMinute
].

//integrity constraint for valid time:
X[invalid] :- X:time, (X.hourOfDay[invalid];
X.minuteOfHour[invalid]; X.secondOfMinute[invalid]).
// a secondOfMinute is represented by an integer
secondOfMinute::integer.
// integrity constraint for valid secondOfMinute:
X[invalid] :- X:secondOfMinute, (X<0; X>59).

// a minuteOfHour is represented by an integer
minuteOfHour::integer.
// integrity constraint for valid minuteOfHour:
X[invalid] :- X:minuteOfHour, (X<0; X>59).

// a hourOfDay is represented by an integer
hourOfDay::integer.
// integrity constraint for valid hourOfDay:
X[invalid] :- X:hourOfDay, (X<0; X>23).

// DATE and TIME

// Concept Definition Date and Time: this represents together a
specific point of time, i.e. an instant
dateAndTime::instant[
date=>date,
time=>time
].
// integrity constraint for valid dateAndTimes:
X[invalid] :- X.date[invalid]; X.time[invalid].

/* some axioms for date / time / date and time */

// equality of a date
X[equal(Y)] :-
Y:date, X:date,
X.dayOfMonth = Y.dayOfMonth,
X.monthOfYear = Y.monthOfYear,
X.year = Y.year.

// test if a given date X is before another date Y
X[before(Y)] :-
(Y:date, X:date),
((X.dayOfMonth < Y.dayOfMonth, X.monthOfYear = Y.monthOfYear,
X.year = Y.year);
(X.monthOfYear < Y.monthOfYear, X.year = Y.year);
(X.year < Y.year)).

// test if a given date X is after another date Y
X[after(Y)] :-
(Y:date, X:date),
((X.dayOfMonth > Y.dayOfMonth, X.monthOfYear = Y.monthOfYear,
X.year = Y.year);
(X.monthOfYear > Y.monthOfYear, X.year = Y.year);
(X.year > Y.year)).

// this is simplified and ignores the leap year
date[daysAfterChrist=>integer].
X[daysAfterChrist->Y] :-
X:date, Z:monthOfYear, X.monthOfYear=Z,
Y is (X.dayOfMonth + Z.daysAfterBeginOfYear + (X.year*365)).

// the difference in days between 2 dates
date[daysUntil(Y)=>integer].
X[daysUntil(Y)->Z] :-
Y:date, X:date,
Z is Y.daysAfterChrist - X.daysAfterChrist.

// ****************** time ******************
// test if two given times are the same
X[equal(Y)] :-
X:time, Y:time,
X.secondOfMinute = Y.secondOfMinute,
X.minuteOfHour = Y.minuteOfHour, X.hourOfDay = Y.hourOfDay.

// test if a given time X is before another time Y
X[before(Y)] :-
(X:time, Y:time),
((X.secondOfMinute < Y.secondOfMinute, X.minuteOfHour =
Y.minuteOfHour, X.hourOfDay = Y.hourOfDay);
(X.minuteOfHour < Y.minuteOfHour, X.hourOfDay = Y.hourOfDay);
(X.hourOfDay < Y.hourOfDay)).

// test if a given time X is after another time Y
X[after(Y)] :-
(X:time, Y:time),
((X.secondOfMinute > Y.secondOfMinute, X.minuteOfHour =
Y.minuteOfHour, X.hourOfDay = Y.hourOfDay);
(X.minuteOfhour > Y.minuteOfhour, X.hourOfDay = Y.hourOfDay);
(X.hourOfDay > Y.hourOfDay)).

// computes the amount of seconds from midnight
time[secondsFromMidnight=>integer].
X[secondsFromMidnight->Y] :-
X:time,
Y is X.secondOfMinute + (X.minuteOfhour*60) +
(X.hourOfDay*60*60).

// the difference in seconds between 2 times
time[secondsUntil(Y)=>integer].
X[secondsUntil(Y)->Z] :-
Y:time, X:time,
Z is Y.secondsFromMidnight - X.secondsFromMidnight.

// ** date and time **
//test if Date and Time are equal
X[equal(Y)] :-
X:dateAndTime, Y:dateAndTime,
X.date[equal(Y.date)],
X.time[equal(Y.time)].

//test if a given date and time X is before another date and
time Y
X[before(Y)] :-
X:dateAndTime, Y:dateAndTime,
(X.date[equal(Y.date)], X.time[before(Y.time)]);
X.date[before(Y.date)].

//test if a given date and time X is after another date and
time Y
X[after(Y)] :-
X:dateAndTime, Y:dateAndTime,
((X.date[equal(Y.date)], X.time[after(Y.time)]);
X.date[after(Y.date)]).

//the difference in seconds between two different DateAndTime
dateAndTime[secondsUntil(Y)=>integer].
X[secondsUntil(Y)->Z] :-
Y:dateAndTime, X:dateAndTime,
Z is Y.time.secondsFromMidnight + Y.date.daysAfterChrist * 24 * 60 * 60 -
(X.time.secondsFromMidnight + X.date.daysAfterChrist * 24 * 60 * 60).

//the difference in (decimal) days between two different
dateAndTime
dateAndTime[daysUntil(Y)=>integer].
X[daysUntil(Y)->Z] :-
X:dateAndTime, Y:dateAndTime,
Z is X.secondsUntil(Y)/60/60/24.

//The current Date manually since we don't have build in
function yet
cURRENT_DATE:dateAndTime[
  date->_:date[dayOfMonth->24,monthOfYear->4,year->2004],
time->_:time[hourOfDay->23,minuteOfHour->40,secondOfMinute->00]
].

*******************************************************************************
// Integrity Constraint to enforce signatures:
// If there is an instance X of Class, that has an Attribute Y and
// there is a signature definition corresponding to Attribute
// then Y has to be an instance of RangeofAttribute
Y:RangeofAttribute :- X:Class, X[Attribute->Y],
Class[Attribute=>RangeofAttribute].

// Integrity Constraint to invalidate instance not
// corresponding to a signature:
// If there is an instance X of Class, that has an Attribute Y and
// there is NO signature definition corresponding to Attribute
// then X is invalid
X[valid(Attribute)] :- X:Class, X[Attribute->_Value],
Class[Attribute=>_Range].
X[invalid] :- X:_Class, X[Attribute->_Value], 
          X[valid(Attribute)].
/*****************************/

The "Purchase" ontology defines general concepts for purchasing a product (there is
a buyer, a seller, a product with a price, a payment method, and delivery).
version -> 1.2

// Concepts

address[
    name => string,
    street => string,
    number => integer,
    zipcode => string,
    city => city,
    state => state,
    country => country
].

buyer[
    shipTo => address,
    billTo => address,
    // Note that the item price given by the buyer denotes a
    price *limit*
    purchaseIntention =>> tradeItem,
    hasPayment =>> paymentMethod
].

seller[
    address => address,
    saleIntention =>> tradeItem,
].

tradeItem[
    product => product,
    price => price
].

// This is the superclass for ticket:
product[
    name => string
].

price[
    amount => real,
    currency => string
].

paymentMethod[
    name => string
].

// A trade is an actual agreement on trading items between two
partners.
trade[
    items =>> tradeItem,
buyer => buyer,
seller => seller,
payment => paymentMethod
].

// Delivery of good as an effect of a purchase
delivery[
    products =>> product,
    receiver => buyer,
    sender => seller
].

// Axioms: purchasing only possible if

// 1) Only products can be traded/delivered if the Buyer requires them (purchaseIntention) and if the seller provides them (saleIntention)
// and if the selling price is not higher than the price intended by the buyer:
T[invalid] :- T:trade[items->>I], tnot T.seller[saleIntention->>I].
T[invalid] :- T:trade[items->>I], tnot priceok(I).
priceok(I) :- T:trade[items->>I], T.buyer[purchaseIntention->>I],
I.product = I1.product,
I.price.currency = I1.price.currency,
I.price.amount =< I1.price.amount.

// 2. Only products can be delivered if
// the receiver requires them (purchaseIntention) and if the sender provides them (saleIntention)
D[invalid] :- D:delivery[products->>P],
tnot D.receiver.purchaseIntention[product->P].
D[invalid] :- D:delivery[products->>P],
D.receiver.saleIntention[product->P].

// Instances (needed throughout the use case modeling)

creditcard::paymentMethod.

creditcard[
    name => string,
    number => string,
    expmonth => month,
    expyear => year,
    // type is something like visa, masterCars, amex, diners, ...
    type => string].

cash::paymentMethod.
cash[
Goals

Goals denote what a user wants to receive when using a Web Service. For all modeling primitives of Goals, there are so-called Goal Templates and Goal Instances. The former describes the general structure of a Goal. Goal Instances instantiate a Goal Template by specifying certain attributes for the which is instantiated by a user for expressing a certain desire. Goal Templates can also be thought of as pre-defined Goals, while Goal Instances are concrete requests for Web Services during runtime.

In the use case, we have one Goal: a user wants to buy a ticket online for a train connection from Innsbruck to Frankfurt on a certain date. The Goal Template states that the desire is to get a train ticket for an itinerary and for a customer, according to the knowledge defined in the ontologies. The Goal Instance specifies concrete values for the template structure. Listing 4 shows this Goal with the following elements:

- **postcondition**: A ticket for a train itinerary from Innsbruck to Frankfurt on May, 23rd 2004, valid for the customer Rubén Lara. The postcondition contains a query which specify a request for a seller that sells the desired ticket.
- **effect**: there is a trade, i.e. a purchase taking place for the ticket with Rubén Lara as the Buyer.

NOTE: The modelling of Goals is under ongoing development. Listing 4 only shows the version existing at the time of writing this version of the Deliverable. The most recent versions of the Goal modeling are available from the Appendix A (In the Appendix, the Goal and the Capability of the corresponding Web Service are provided in one single file in order to illustrate Web Service Discovery in FLORA-2).

Listing 4: Goal - buying a train ticket online

```plaintext
// Goal

// Non-functional Properties (Core Properties)
title -> "Goal: buying online train ticket",
creator -> "DERI",
contributor -> "Rubén Lara, Holger Lausen, Axel Polleres, Michael Stollberg",
date-> 20040418,
type-> "WSMO Goal",
identifier ->
```
http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/goal.flr
identifier ->
http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/goal.flr
version -> 1.10

// usedMediators
// OO Mediator for using ontologies
VTA-OOM-Goal1

// Goal Templates

// Goal Postcondition Template

myTicket:ticket[
  itinerary->X1,
  customer->X2
].

// Goal Effect Template

myTrade:trade[
  items->>_#:tradeItem[
    product->myTicket
    //Price limit is left open
  ],
  buyer->X1
].

// ***********************************
// data for GOAL Postcondition
// ***********************************
// Instantiated Goal:
?-
myTicket:ticket,
myTicket.itinerary.startLocation=innsbruck,
myTicket.itinerary.endLocation=frankfurt,
myTicket.itinerary.departure=MyDepartureDate,
MyDepartureDate:dateAndTime,
MyDepartureDate.date.dayOfMonth=17,
MyDepartureDate.date.monthOfYear=3,
MyDepartureDate.date.year=2004,
MyDepartureDate.time.hourOfDay>=17,
MyDepartureDate.time.hourOfDay=<19,
myTicket.customer.name=MyAddress.name,

// ***********************************
// data for GOAL Effect
// ***********************************
// in flora2 implementation effect and poscondition are
// concatenated
MyAddress:address,
Web Services

For our use case we define one Web Service: an (imaginary) online train ticket booking services for international train itineraries, offered by the Austrian national train operator ÖBB. Of course, this Web Service can be split up into several Web Services wherefore technologies for composition and Orchestration would be needed. But as a our intention within the current version of this use case modeling is to test and showcase the basic modeling of Web Services, we restrict ourselves to only one Web Services at this point in time.

As the referenced version of WSMO does only provide specifications for the description elements for Web Service Capability modeling, we restrict the Web Services models to the Capabilities at this point in time. A Web Service Capability in WSMO is described by pre- and postconditions, assumptions and effects. The primary information for suitability of a Web Service for satisfying a given Goal is the postcondition (the Web Service postcondition has to logically satisfy the Goal...
postcondition, which is the core of the discovery mechanisms). The other description elements are secondary information for determining suitability, i.e. filtering the set of Web Services that potentially match the Goal. More detailed discussion of the Discovery mechanism of WSMO Goals and Capabilities is provided in section 3.1.3.

Listing 5 shows the Capability specification for the Web Service. Here, we only specify the postcondition and the effect of the Capability as these are the most relevant information for Web Service Discovery. The elements specify the following:

- **postcondition**: states that the Web Service is a Seller with a SaleIntention for train ticket in Austria and Germany.
- **effect**: there is a trade, i.e. a purchase taking place for the ticket with the Web Service owner as a seller and a Buyer that accepts creditcard as payment method.

NOTE: The modelling of Goals is under ongoing development. Listing 5 only shows the version existing at the time of writing this version of the Deliverable. The most recent versions of the Web Service Capability modeling are available from the Appendix A (In the Appendix, the Goal and the Capability of the corresponding Web Service are provided in one single file in order to illustrate Web Service Discovery in FLORA-2).

Listing 5: Capability of ÖBB Web Service for Booking Online Train Tickets for Austria and Germany

```plaintext
// Web Service Capability

// Non-functional Properties (Core Properties)
title -> "Web Service Capability: selling online train tickets for Austria and Germany"
creator -> "DERI"
contributor -> "Rubén Lara, Holger Lausen, Axel Polleres, Michael Stollberg"
date-> 20040430
type-> "WSMO Web Service Capability"
identifier -> http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/goal.flr
version -> 1.10

// usedMediators
// OOO Mediator for using ontologies
VTA-OOM-WS1Cap

// **********************************************
// CAPABILITY postcondition
// **********************************************
?- OBBTicket:ticket,
//Start location have to be in Austria or Germany
(OBBTicket.itinerary.startLocation.country=austria; OBBTicket.itinerary.startLocation.country=germany),
```
// End location have to be in Austria or Germany
(OBBTicket.itinerary.endLocation.country=austria;
OBBTicket.itinerary.endLocation.country=germany),
// Departure has to be at least one hour later than current date and time
// currentDate.secondsUntil(OBBTicket.itinerary.departure) > 3600,

// ***********************************
// Capability Effect
// ***********************************
// We concatenate effect and poscondition in one
// query for the moment...
OEBBTrade:trade,
OEBBTrade..items.product=OEBBticket,
// Do we have to specify this here? In fact the concrete data of seller oebb,
// should be provided by an oebb-ontology, which contains all the facts
// on OEBB, or no?
OEBBTrade.seller=oebb,
OEBBTrade.seller..saleIntention.product=OBBTicket,
OEBBTrade.buyer..acceptsPayment:creditCard,
{
(OEBBTrade.buyer..acceptsPayment.expyear>=currentDate.date.year,
OEBBTrade.buyer..acceptsPayment.expmonth>=currentDate.date.monthOfYear);
(OEBBTrade.buyer..acceptsPayment.expyear>currentDate.date.year)
}.

/*******
* A rasonar now would have to chck if the intersection of both
* descriptions (goal and capability) is satisfiable
* some example instance to check if there is an intersection
*******/

mytrade:trade[
  items->>_ti:tradeitem[
    product->_ticket:ticket[
      itinerary->_#:itinerary[
        startLocation->innsbruck,
        endLocation->frankfurt,
        departure->_#:dateAndTime[
          date->_#:date[
            dayOfMonth->17,
            monthOfYear->3,
            year->2004
          ],
          time->_#:time[
            hourOfDay->17,
            minuteOfHour->20,
            secondOfMinute->20
          ]
        ]
      ]
    ]
  ]
]
Mediators

As requirements for our use case, we have identified the need for 3 types of Mediators:

- OO-Mediators which connect the specific WSMO components and provide the needed ontology integration facility as a mediation service.
- WG Mediators for connecting the Goals and Web Services after successful discovery of suitable Web Services for searching valid train itineraries and online booking of tickets for an train itinerary.

In the following we model the concrete mediators needed for the use case. These Mediators are applied by the different components described before using the “usedMediators” modeling element.
**OO-Mediation**

OO Mediators "connect" the ontology/ies with the component that is using it. According to the WSMF/O-principle of strong de-coupling, all issues related to integrate (meaning to resolve all namespace and access issues) and mediate (meaning to resolve heterogeneities) the ontologies to be used are located in the OO Mediator. The user who uses a OO Mediator with the "usedMeditaors"-tag should therefore not care about which ontology he refers to - he just gets his "Information Space" by the OO Mediator and uses it.

In order to import the ontologies into all our components, we need 3 OO Mediators:

1. importing the Date and Time Ontology into the Train Connection Ontology
2. importing all ontologies in to the Goal
3. importing all ontologies in to the Web Service Description, here the Capability

Listings 6-8 specify these 3 OO Mediators. The general structure of an OO Mediator is that the sourceComponent is the imported ontologies (can also be other OO Mediators), and the targetComponent is the user of the imported ontologies. The internal structure and functionality of a OO Mediator is not stable specified yet, thus we only specify the connection facility in the Listings. We do not provide the OO Mediators as FLORA-2 resources for download because the connection facility has to be adopted to the Flora-module technology.

<table>
<thead>
<tr>
<th>Listing 6: OO-Mediator 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>// OO Mediator: importing the Date and Time Ontology into the Train Connection Ontology</td>
</tr>
<tr>
<td>// non-functional properties</td>
</tr>
<tr>
<td>title -&gt; &quot;OO Mediator: importing the Date and Time Ontology into the Train Connection Ontology&quot;</td>
</tr>
<tr>
<td>creator -&gt; &quot;DERI&quot;</td>
</tr>
<tr>
<td>contributor -&gt; &quot;Michael Stollberg&quot;</td>
</tr>
<tr>
<td>date-&gt; 20040418,</td>
</tr>
<tr>
<td>type-&gt; &quot;WSMO OO Mediator&quot;,</td>
</tr>
<tr>
<td>identifier -&gt; <a href="http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/VTA-OOM-trainConnection">http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/VTA-OOM-trainConnection</a>,</td>
</tr>
<tr>
<td>version -&gt; 1.0</td>
</tr>
<tr>
<td>// Source and Target Component</td>
</tr>
<tr>
<td>sourceComponent -&gt;&gt; {dt.flr}, // identifier of &quot;Date and Time Ontology&quot;</td>
</tr>
<tr>
<td>targetComponent -&gt;&gt; {tc.flr}, // identifier of &quot;International Train Ticket Ontology&quot;,</td>
</tr>
<tr>
<td>// Mediation Service not needed here</td>
</tr>
</tbody>
</table>
### Listing 7: OO-Mediator 2

```
// non-functional properties
// title -> "OO Mediator: importing all ontologies in to the Goal"
// creator -> "DERI"
// contributor -> "Michael Stollberg"
// date-> 20040418
// type-> "WSMO OO Mediator"
// identifier ->
// http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/VTA-OOM-Goal1
// version -> 1.0

// Source and Target Component
sourceComponent ->> {dt.flr, po.flr}, // identifier of needed ontologies
targetComponent ->> {goal.flr}, // identifier of Goal

// Mediation Service not needed here
```

### Listing 8: OO-Mediator 3

```
// non-functional properties - only selection
// title -> "OO Mediator: importing all ontologies in to the Web Service Description, here the Capability"
// creator -> "DERI"
// contributor -> "Michael Stollberg"
// date-> 20040418
// type-> "WSMO OO Mediator"
// identifier ->
// http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/VTA-OOM-WS1Cap
// version -> 1.0

// Source and Target Component
sourceComponent ->> {dt.flr, po.flr}, // identifier of needed ontologies
targetComponent ->> {capability.flr}, // identifier of Web Service Capability,

// Mediation Service not needed here
```

**WG-Mediators**
A WG Mediator denotes the differences between a Goal and a Web Service
Capability in order to make them matching. The difference is stated in a reduction. In fact, the reduction defines the intersection between the information space of the Goal and the information space of Capability. Thereby, only such information will be interchanged between the Goal and the Web Service that are valid, meaning that with all outputs, postconditions, and effects of the Web Service the Goal is satisfiable.

In our use case, we only need 1 WG Mediator for connecting the Goal and the Web Service, specified in Listing 9. The WG Mediator, as all other WSMO components, has to be aware of the ontologies used in the components to be connected. Therefore, it uses the OO Mediators defined for the Goal and for the Web Service. In the use case, these OO Mediators are similar - but the rule of using the OO Mediators of the WSMO components to be connected holds in the general case as well (also for all other types of WSMO Mediators that use OO Mediators: GG Mediators and WW Mediators). The Reduction in the WG Mediator restricts the set of values for the Goal and the Web Service to those for which the usage of the Web Service is valid for satisfying the Goal. More precisely, the Reduction is the intersection of valid knowledge items of the Goal and of valid items of the Web Service.

---

Listing 9: WG-Mediator between Goal and Web Service

```plaintext
// OO Mediator: importing all ontologies in to the Web Service
Description, here the Capability

// non-functional properties - only selection
title -> "WG Mediator"
creator -> "DERI"
contributor -> "Michael Stollberg"
date-> 20040418
type-> "WSMO WG Mediator"
identifier ->
http://www.wsmo.org/2004/d3/d3.2/v0.1/20040419/resources/VTG-WGM1
version -> 1.0

// Source and Target Component
sourceComponent ->> {goal.flr} // identifier of Goal
targetComponent ->> {capability.flr} // identifier of Web Service Capability,

// usedMediators
// The WG Mediator uses the OO Mediators of the source and the
target component in order to use the terminology
VTA-OOM-Goal1
VTA-OOM-WS1Cap

// Reduction to be specified
```

**GG-Mediators**

A GG Mediator connects Goals by specifying a reduction between them. There is no GG Mediator needed in the use case.
**WW-Mediators**
A WW Mediator connects Web Services used by another Web Service in their Orchestration, resolving heterogeneities at all levels (data, process, protocol). There is no WW Mediator in this use case.

**3.1.3 SWS Mechanisms based on WSMO component models**

On basis of the models for the WSMO components specified above, we can define the following automated mechanisms: Web Service Discovery, Web Service Composition, and Web Service Execution. In the following we explain how these mechanisms work and which parts of the WSMO models they use.

**Web Service Discovery**

Web Service Discovery is concerned with inference-based mechanisms that detect suitable Web Service for a given Goal. This means that the discovery mechanism searches available Web Service descriptions and determines whether these can be used to fulfill a certain Goal. The overall structure of WSMO supports Web Service discovery explicitly by introducing the notions of Goals and Web Services. The requirements and the approach for Web Service Discovery in WSMO is exhaustively discussed in [Keller et al., 2004]. Here, we shortly summarize the most important aspects and explain how the discovery mechanism works in the use case presented.

The functionality of the discovery mechanism can be separated into three major aspects:

1. **The Core: Goal-Capability-Matching**
   Goal-Capability matching determines whether the Capability of a Web Service Description can satisfy the given Goal, i.e. if the Web Service can be used for solving the Goal. Therefore, it has to be proven that the Capability logically entails the Goal with the premise that the conditions for successful usage of the Web Service have to be fulfilled. We discuss this in more detail below.

2. **Heuristics for establishing Goal-Capability Matching**
   Goal-Capability matching is only successful, i.e. it returns a set of suitable Web Services to solve a given Goal, if the Goal and the Web Service Capability match perfectly. This means that for all postconditions and effects of the Web Service the Goal is satisfied. This might not hold for many cases, as their might be differences between the Goal and the Capability; but a Web Service might be usable for solving Goal when the valid set of outputs of the Web Service is restricted - also vice versa, i.e. the Web Service can be used to solve a Goal within certain restrictions on the Goal.

   In WSMO, these differences are explicitly stated in a WG Mediator. This restricts the valid values between a Goal and a Web Service Capability, thereby ensures Goal-Capability-Matching between Goals and Capabilities that only match partly, and thus broadens the set of possible usable Web Services for solving a Goal. For determining the required reduction in a WG-Mediator, specific heuristics can be applied.

3. **Filter mechanisms for improving discovery results**
   The Discovery mechanism in general returns a set of suitable Web Services for solving a Goal. In order to improve the quality of the results set, additional filter mechanism can be applied. These can be based on the non-functional
properties of Web Services, or can take some preferences of the customer into account.

With regard to the WSMO models of the use case defined above, we can now show how the heart of Web Service Discovery in WSMO, i.e. the Goal-Capability-Matching works. Therefore, we have to show that the Proof Obligation for Goal-Capability-Matching holds for the models of the Goal and the Capability defined in the use case. The Proof Obligation for Goal-Capability-Matching is defined as follows:

\[
PO_{\text{gon}}(G, C) \equiv \{ O_G, O_C, M_C, M_G \} \models \exists \chi, \ldots, \chi_n: (\psi_{\text{pre}} \land \psi_{\text{ass}}) \land (\psi_{\text{post}} \rightarrow \phi_{\text{post}}) \land (\psi_{\text{eff}} \rightarrow \phi_{\text{eff}}))
\]

This Proof Obligation states that under consideration of all Ontologies and Mediators used in the Goal and the Capability description (2nd line), the user is able to provide concrete values for the input parameters of a service, such that the Preconditions defined in the Capability (3rd line) is satisfied, the Capability postconditions imply the Goal postconditions and the Capability effects imply the Goal effects.

// to be updated

In the use case, we have 1 Goal (see Listing 4) and 1 Capability (see Listing 5). Obviously, the Proof Obligation is fulfilled in this case because the Goal and Capability are homogenenous. In fact, the Capability postcondition is true because there exists an instance in the information space that fulfills the postcondition (the instance is a ticket for the itinerary form "Innsbruck" to "Frankfurt" - as pre-defined instances of Location, wherein the startLocation is in Austria, the EndLocation is in Germany, and the Departure is more than 1 hour later than the current date and time). Also, this ticket instance fulfills the desired Goal. The same holds for the effects of the Capability and the Goal. The Proof Obligation says that if the Capability postcondition is true and the Goal postcondition is true (similar for the effects), than the Goal matches the capability. In this case, the Goal and the Capability match perfectly. This means that there is no Reduction needed in the WG Mediator, and thus no heuristics needed for determining the requirements for reduction.

Note that in the current example, the query-parts of the Capability and the Goal description describe the "requests for a partner"; actually, these queries describe the functionality of the Discovery Mechanism. Imagine that these are taken out of the Goal and Capability descriptions and integrated into the discovery mechanism, this seems to be a feasible, generalizable approach for Web Service Discovery within WSMO.

**Web Service Composition**

[not in this version]

**Web Service Execution**
3.1.4 Conclusions

We have described a real-world setting of using Semantic Web Services for a Virtual Travel Agency (VTA) that provides an end-user service for booking international train tickets, thereby aggregating Web Services of different e-Tourism Service Providers. The set up of this use case and the system architecture of the VTA here is conform to the general structure of the VTA use case described in Section 2.1.

Within the WSMO models defined in this use case we have shown how to model the different components of WSMO, with regard to the stable WSMO modeling elements available at this point of time:

- Ontologies: all needed domain knowledge is provided in the ontologies.
- Goals: We have specified 2 Goals in the Use Case, according to the real-world settings of searching and booking international train tickets
- Web Services: We have modeled the Capabilities of different Web Services needed for the Use Case
- Mediators: We have modeled the needed WSMO Mediators needed for the Use Case, namely OO Mediators, WG-Mediators, and WW Mediators.

Furthermore, we have outlined the general workflow of the WSMO Discovery mechanism that works on the WSMO models for Goals and Capabilities.

The outcome of the first use case modeling are manifold. First of all, it shows how the different WSMO components are modeled concretely. This gives answers to many questions that have been arising within WMSO: a more concrete understanding of Goals in WSMO and what they actually express, what is defined in a Web Service Capability with special regards to the difference between preconditions and assumptions, postconditions and effects respectively, and the concept of Mediators in WSMO. Further major outcomes of the use case and testing efforts so far is a concrete specification of how to model the different types of axiom definitions in WSMO, as well as further insights in the discovery mechanism that works on the WSMO models for Goals and Web Service Capabilities. The scope of the use case is restricted to the most essential building blocks of WSMO at this point of time, and it will be updated and extended in the future for testing and showcasesing further WSMO constructs.

3.2 B2B Integration with Semantic Web Services

[not in this version]

3.2.1 Functional Requirements

similar to 3.1.1

3.2.2 WSMO Modeling

similar to 3.1.2
Ontologies

Goals

Web Services

Mediators

3.2.3 SWS Mechanisms based on WSMO component models

similar to 3.1.3

3.2.4 Conclusions

similar to 3.1.4

4. Conclusions and Further Work

Appart from discussing possible usage scenarios of Semantic Web Services, the major interest in this deliverable is to test and verify WSMO modeling for recursive development of WSMO, and to serve as a testbed for development of WSMO-based technologies. The deliverable is intended to exemplify and showcase the usage of WSMO for modeling different aspects related to Semantic Web Services, and it will continuously be updated according to further development of WSMO.

According to the current status of WSMO, the most interesting aspects are:

- to showcase concrete modeling of a real-world scenario in WSMO
- thereby gain a better understanding of WSMO, its distinct components, and how these are related
- a more precise definition of specific elements (esp. axioms), their meaning and how to model them

The major outcome of the Use Case modeling provided in this deliverable are:

- exemplification of modeling the core components in F-Logic according to the current specification of WSMO modeling elements
- understanding and specification for modeling different types of axioms in WSMO
• insights on the requirements and workflow of the WSMO Discovery mechanism, i.e. for matching Goals and Web Service Capabilities

The directions for future work in this deliverable are:

• enhance the scope of the Use Case modeling.
• provide complete modeling of other use cases, esp. for the "B2B Integrating with Semantic Web Services" as described in Section 2.2
• update the WSMO models according to new developments or changes in the WSMO Specification

References


Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, and SWWS; by Science Foundation Ireland under the DERI-Lion project; and by the Austrian government under the CoOperate programme.

The editors would like to thank to all the members of the WSMO working group for their advises and inputs to this document.

Appendix A: Flora2-F-Logic for the VTA - Use Case

Here, the complete WSMO models for the VTA Use Case described in Section can be download as computational resources for testing and development of WSMO. The files below contain the WMSO models of the use case in Flora2-compatible syntax. For testing and development, the files can be loaded into different Flora modules.

Information and download of Flora2 is provided here.

NOTE: the WSMO models for Flora2 are currently under construction. The most recent resources can be accessed via CVS web-interface at: http://cvs.deri.at/cgi-bin/viewcvs.cgi/wsmo//d3/d32/resources/.

Ontology 1: "International Train Connections Domain Ontology"
Ontology 2: "General Date and Time Ontology"
Ontology 2: "Purchase Ontology"

The models for the Goal and the Web Service Capability are provided in one file (goal.flr) at this moment - only for the ease of use for testing and development purpose. If you download this file into the same directory as the ontology files above, you can test and examine the models in FLORA-2.