



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

### WSMO Working Draft 28 June 2004

**This version:**

<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/>

**Latest version:**

<http://www.wsmo.org/2004/d3/d3.2/v0.1/>

**Previous version:**

<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/>

**Editors:**

Michael Stollberg  
Holger Lausen  
Axel Polleres  
Rubén Lara

**Authors:**

Michael Stollberg  
Holger Lausen  
Axel Polleres  
Rubén Lara  
Uwe Keller  
Michal Zaremba  
Dieter Fensel  
Michael Kifer

**Reviewer:**

Christoph Bussler

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

Copyright © 2004 [DERI](#)®, All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

---

## Abstract

This document exemplifies the usage of the Web Service Modeling Ontology WSMO for modeling possible Web Service driven applications. We first outline general usage of Semantic Web Service in different application fields, identifying the usage

scenarios and the arising technical requirements. Then, we provide the WSMO modeling of specific use cases, in order to demonstrate WSMO modeling and for supporting 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 [[Roman et al., 2004](#)].

## Related Documents

WSMO Standard: [D2 v0.2 Web Service Modeling Ontology \(WSMO\)](#), last version at: <http://www.wsmo.org/2004/d2/>

WSMO Primer: [D3.1 v01. WSMO Primer](#)

WSMO Reasoning: [D5.1 v01. Inferencing Support for Semantic Web Services: Proof Obligations.](#)

---

## 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](#)

[2.1.4 Usage Scenarios](#)

[2.1.5 System Architecture](#)

[2.2 B2B - Integration with Semantic Web Services](#)

[2.2.1 Description](#)

[2.2.2 Scope](#)

[2.2.3 Actors, Roles and Goals](#)

[2.2.4 Usage Scenarios](#)

[2.2.5 System Architecture](#)

### 3. WSMO Use Case Modeling

[3.1.VTA for International Online Train Ticket](#)

[3.1.1 Use Case Overview](#)

[3.1.2 WSMO Modeling](#)

[Ontologies](#)

[Goal](#)

[Web Services](#)

[Mediators](#)

[3.1.3 SWS mechanisms based on WSMO models](#)

[3.1.4 Conclusions](#)

### 4. Conclusions and Future Work

### References

## Acknowledgements

[Appendix A: Flora2-F-logic for the VTA - Use Case](#)

[Appendix B: Change Tracking](#)

[Appendix C: Review Comments](#)

---

# 1. Introduction

This document exemplifies the usage of the Web Service Modeling Ontology WSMO for describing Semantic Web Services and related constructs. To this end, we describe two possible use cases of Semantic Web Services in B2C and B2B scenarios and showcase how these can be modeled in WSMO. We briefly replicate the objectives and the approach of WSMO and outline how specific use cases can be modeled in WSMO along with explanations on the modeling decisions. The modeling is exemplified in a human readable syntax for WSMO descriptions. Besides, resources in pure F-Logic generated from these descriptions which aim at solving Web service discovery based on WSMO using an engine such as FLORA-2 are provided in the appendix.

This Deliverable is intended to evolve in accordance to the ongoing development of WSMO itself, serving as input and providing valuable insight for testing and adapting the modeling constructs provided in WSMO in real world scenarios for Web Services. So, besides demonstrating how to model Web Services in WSMO, the use cases also allow us to demonstrate the adequacy of our approach in terms of providing an exhaustive framework for covering all relevant aspects of semantic description of Web Services. In the long 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](#) summarizes the objectives and approach of WSMO; [Section 2](#) discusses possible application areas of Semantic Web Services. [Section 3](#) provides the modeling of real world use cases in WSMO. [Section 4](#) concludes the document. The complete WSMO models as computational resources are provided in the [Appendices](#). Besides, we provide facilities for readers to keep track of improvements and discussion related to this document: a Change Tracker in [Appendix B](#) explicitly list the major changes between different versions of this document in order to facilitate readers following the improvements, and [Appendix C](#) lists the comments given by reviewers as well as the actions undertaking to resolve open issues.

## 1.1. Semantic Web Services

A Web Service is a piece of software accessible via the Internet. 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 and interoperability of Web Services.

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 resource that can be published, discovered, composed and executed across the Web in a task driven automatic way” [Arroyo et al., 2004]. In the end, by machine-processable descriptions of the relevant information of Web Services, the following tasks shall be addressed:

- **Automatic Web Service Discovery:** finding Web Services that can solve a goal defined by a service requester, i.e. a desire that a user wants to get resolved automatically by using Web Services.
- **Automatic Web Service Composition:** assembly of services based on their functional specifications in order to provide a functionality on a higher level.
- **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 the WSMO project 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 needed for a full coverage framework for Semantic Web Services are defined (see Figure 1):

**Ontologies** provide the formal semantics of the information used by all other components.

**Goals** specify objectives that a client may have when it consults a web service.

**Web Services** exposed descriptions of the functionality of a Web Service for supporting automated discovery, composition, and execution (called “Capabilities” in WSMO). For supporting automated usage, composition, and execution of Web Services, particular information on the external visible behavior of a Web Service are specified (called “Interface” in WSMO), including information on the technical accessibility and the actual message exchange of Services.

**Mediators** 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.

Further details on the components of WSMO along with exhaustive explanations are presented in the WSMO Primer [Arroyo and Stollberg, 2004].

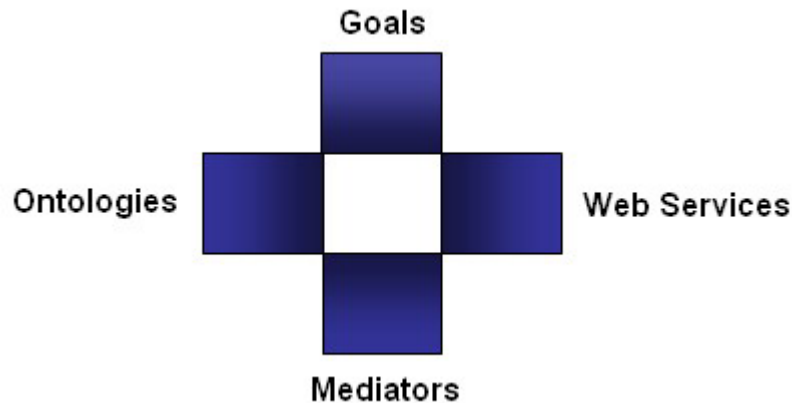


Figure 1. WSMO Components

## 2. Use Cases

Semantic Web Services can be used in manifold application fields. In accordance to the use cases defined in Web Services Architecture Usage Scenarios by the [W3C Web Services Architecture Working Group](#) (see [He et al., 2004]), we discuss two of the most commonly used scenarios to exemplify the usage of SWS technologies in this document:

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. a third party provides a service to end users acting as a Client aggregating other 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 them by the requirements arising for Semantic Web Services technologies. The aspects considered for our use case definitions are as follows:

- **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 requester and one or more services, which achieve measurable results for the requester", and a [usage scenario](#) as "... an atomic step in a path through a use case", i.e. an activity that has to be 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 scenarios by the following informations:

- participating actors and their goals
  - activities to be performed
  - technological requirements
  - 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.

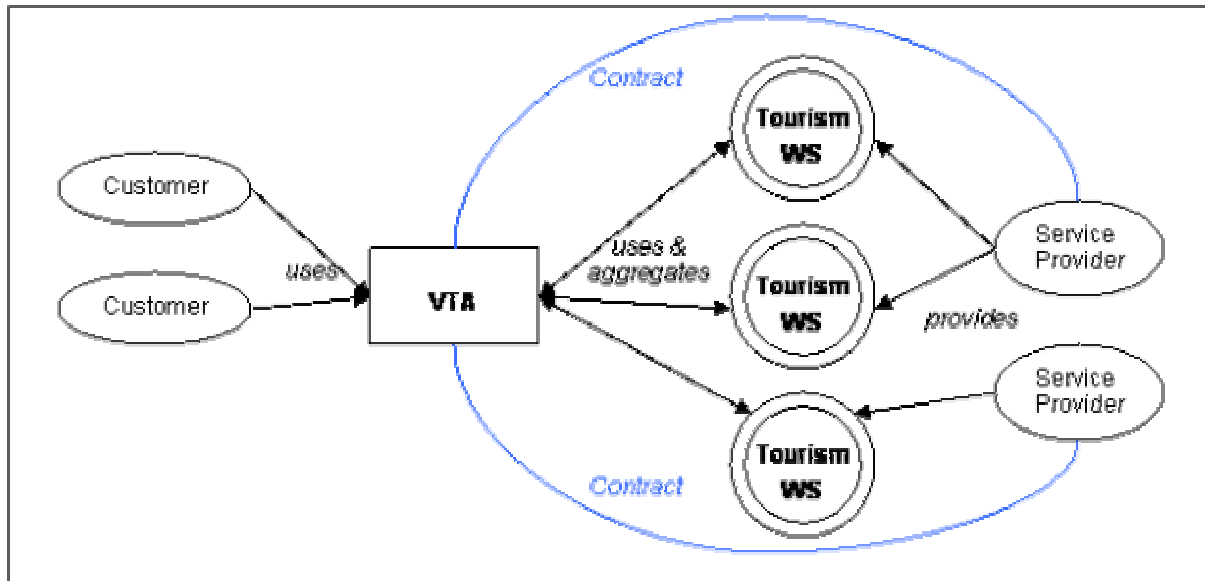
## 2.1 B2C - Virtual Travel Agency

In [He et al., 2004], 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, in the first WSMO use case we will describe a Virtual Travel Agency example that involves automated discovery of Web Services.

### 2.1.1 Description

Imagine a “Virtual Traveling Agency”, called VTA for short, which is an end user platform providing eTourism services to customers. These services can cover all kinds 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 at this point mostly comprise simple information portals along with some web-based customer services (see for example an eTourism service provider in Austria at: <http://www.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 in a (semi-)automatic fashion. Such VTAs providing automated eTourism services to end users thus tremendously enhance the functionality of currently existing VTAs.

Our VTA use case that aggregates Web Services of different tourism service providers in a nutshell shall provide the following functionality: A customer uses the VTA service as the entry point for his requests. These requests must fit to end-user services 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 gives an overview (modified and extended from [W3C Travel Agent Use Case overview](#), as defined in [He et al., 2004]).



**Figure 2. Use Case Overview: Virtual Travel Agency based on Semantic Web Services**

### 2.1.2 Scope

The scenario outlines 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 services for providing an end-user service for booking complete holiday trips worldwide.

We provide the modeling of one such VTA use case in Section [3.1.VTA for International Online Train Ticket](#).

### 2.1.3 Actors, Roles and Goals

In the general use case there are 3 actors. The following defines why they participate in this use case (goal) and the particular interactions they are involved in (roles).

1. **Customer:** the end-user that requests a 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 Providers:** a commercial companies 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 Providers. It provides high-quality tourism services to customers by aggregating the separate services provided by the single Service Providers.
  - *Goal*: provide high-quality end-user tourism services, uses existing tourism services and aggregates these 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 direct human customers interaction or an 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 requirements are modeled in the 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 and queries suitable Web Services and displays results to Customer*
  - **Participating Actors**: Customer, VTA, Tourism Service Providers
  - **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 the "Search" functionality that can be requested by a Customer
    - (2) the Tourism Service Providers' 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 provide mechanisms for automated Service Discovery
  - **Possible Extensions**:
    - the Customer specifies its request in natural language and the request is translated into machine readable form and processed by the VTA service automatically
    - several Web Services are aggregated for providing the requested functionality if it cannot be fulfilled by a single service
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, VTA, Tourism Service Providers
  - **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 and composes them into the functionality to satisfy the user request

(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 credit card number for payment)

- **Technological Requirements:** contract information information requirements are modeled in the 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 an 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, Tourism Service Providers

- **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:

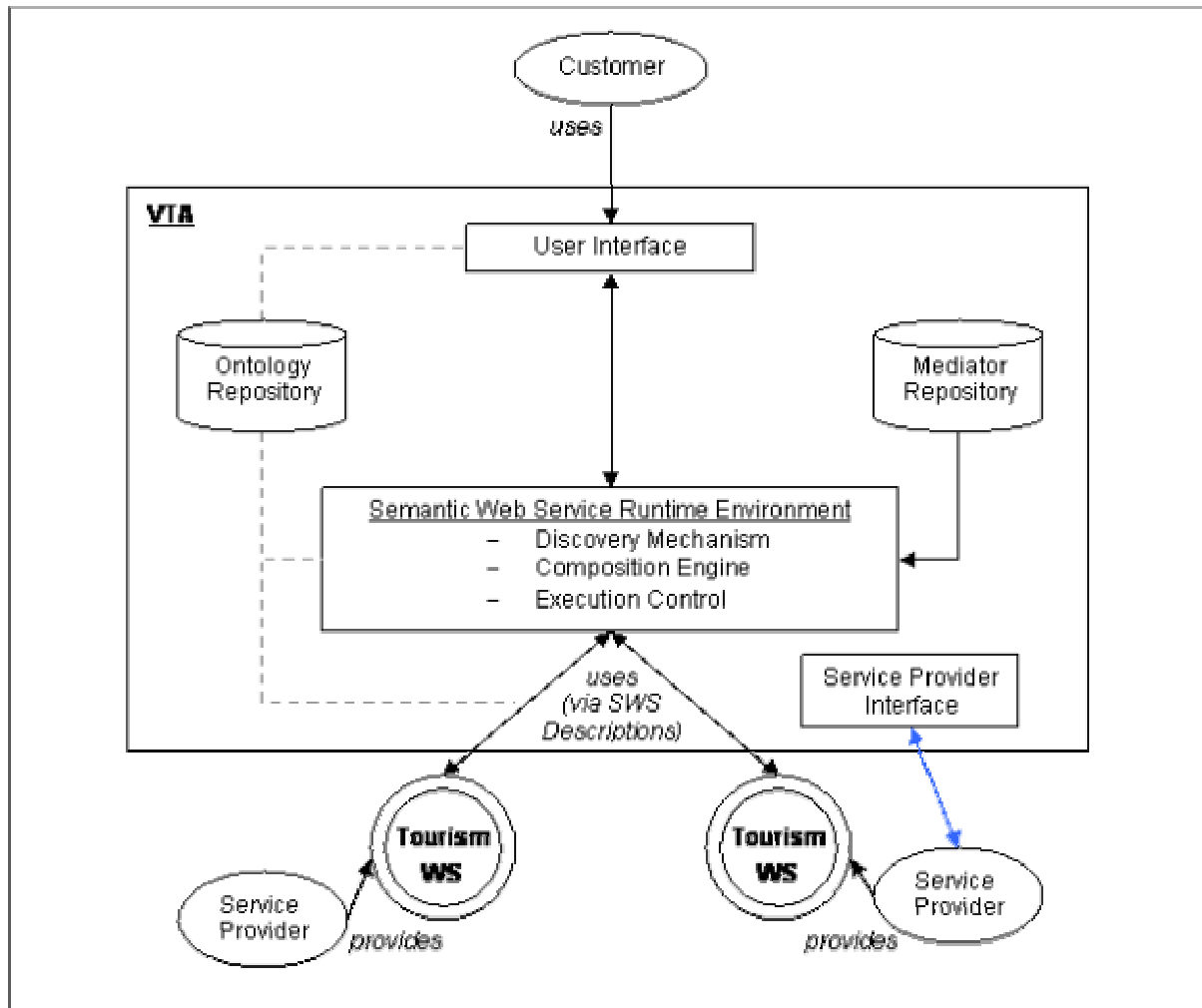
- customer notification
- VTA-Service Provider interaction on accounting and good delivery mandate

## 2.1.5 System Architecture

In this use case, the VTA is the central point of interaction between the Customer and other Web Services. Regarding the technological requirements, it is 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 provide 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 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 properly with heterogeneous resources, thus allowing for appropriate mediation facilities.
- It has to provide interfaces 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 client/server model. In order to support coherent functionality of the VTA and to ensure that the descriptions of Web Services are compatible to this, an overall framework for SWS technologies is needed. This is provided by WSMO.

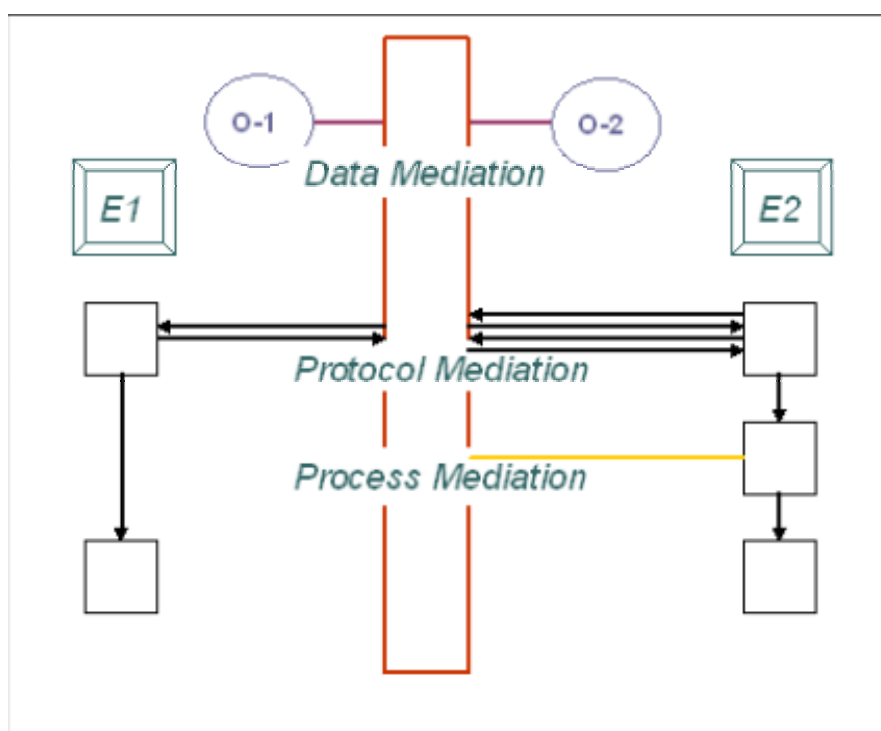
## 2.2 B2B - Integration with Semantic Web Services

The second use case is concerned with the integration of possible heterogeneous resources in B2B settings which is considered as one of the most important application fields of the 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. Assuming that partners may not know each other before, contract negotiation and contract agreement are essential aspects of this use case. The contract agreement defines roles of enterprises in the

conversation, for instance, one of the enterprise E1 becomes the seller and the second enterprise E2 becomes the buyer. An agreement also predefines the order of the messages interchanged between the parties, e.g. the buyer first sends purchase order (PO) and after that it receives purchase order acknowledgement (POA). In contrast to the previous B2C use case, where the client/server model of interactions has been adopted, here partners are equal in the interaction, i.e. a peer-to-peer model is assumed in this use case. Each of the companies has an own set of web services for exchanging business documents electronically. The 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), as well as linking web services and goals. The infrastructure also supports the execution of the contract to fulfill approved agreement.



**Figure 4. B2B Integration with Semantic Web Services**

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**, **heterogonous** 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 **heterogonous**, 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, which represent two 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 business partner to achieve given goal by establishing new 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. Lower level acknowledgments messages for each of the PO and POA can be also exchanged.
  - **Technological Requirements** - The technology should enable conversation between business partners, supporting different process models to achieve given task e.g. buying a product. For example system of one business partner might require a synchronized confirmation for each business document send out, while the system of the other business partner assumes that once the document is send, it does not have to be confirmed. The SWS platform should provide appropriate process mediation mechanism to resolve this issue.
  - **Possible Extensions** - the system of one of the business partner might failed and drop in the middle of conversation (e.g. it receives PO, but never sends a POA). The SWS platform, similarly to workflow engines, takes care to recover from deadlock and livelock errors.
3. *SWS infrastructure crashes - once it recovers, it reliable commence its operations*
  - **Participating actors** - buyer and seller
  - **Activities** - Because of some internal (e.g. lack of power supply for the server) or external (e.g. lack of network connection) failure, the SWS system becomes temporary unavailable. Once it is back online, it commence from the point where the execution has been dropped. None of the messages are lost, none of the processes are executed from the beginning.
  - **Technological Requirements** - Reliable and event driven architecture.
  - **Possible Extensions** - The SWS infrastructure informs all interested parties that it is back online.
4. *E1 and E2 want to deploy a new integration definition type (described in WSMML). 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** - Any new integration type can be compiled and deployed by SWS infrastructure.
- **Technological Requirements** - Standard interface which allows carrying conversation between SWS infrastructure and WSML editor. New integration definition types can be saved and retrieved from the system.
- **Possible Extensions** - Public interface which enables any external party to provide own definitions.

## 2.2.5 System Architecture

The Web Services Modeling Execution (WSMX) is the infrastructure is a WSMO-reference implementation that addresses this use case, see WSMX homepage at: <http://www.wsmo.org/wsmx/>. Therein, a WSMX-platform is 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.

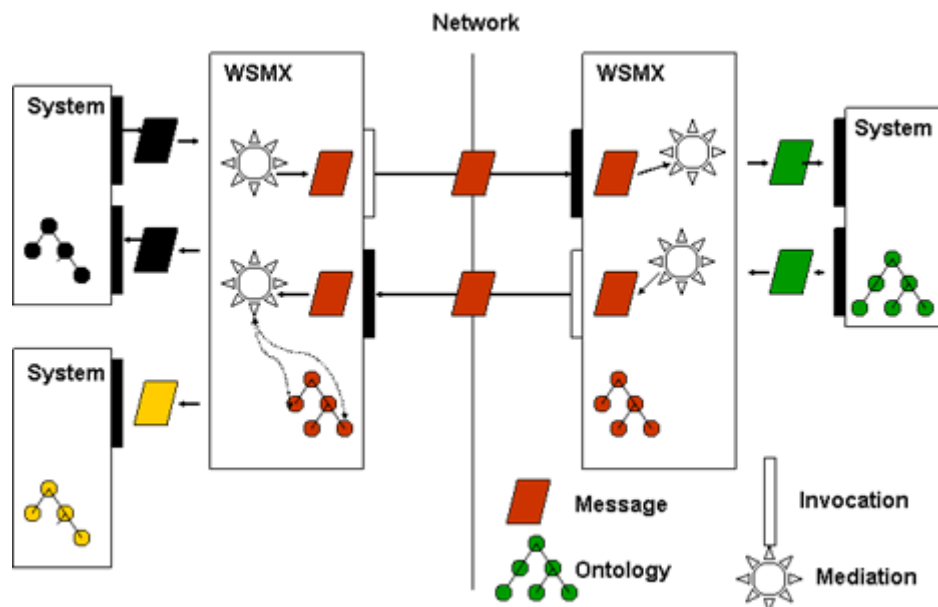


Figure 5. B2B Use Case System Architecture

## 3. WSMO Use Case Modeling

The following exemplifies the usage of WSMO for describing Semantic Web Services for some specific aspects of the use cases described above. In the following we will:

1. describe the specific use case
2. give an overview of how to model the use case in WSMO (property description of specific WSMO components)
3. provide the detailed models in WSMO (using WSML)

4. provide the equivalent of all WSMO models in FLORA-2 syntax in order to allow testing and development in an execution environment

The provided listings use the conceptual model presented in [WSMO Standard, V0.2](#) as the latest stable version of WSMO. During the elaboration of this use, several conceptual refinements have been undertaken which we explicitly describe in the text. For modeling, we apply the syntax defined in in [D16.1 v02 BNF Grammar for WSML language](#). Currently, we support FLORA-2 as an F-Logic reasoner (see [documentation and download of the latest distributions at the FLORA-2 homepage <http://flora.sourceforge.net/>](#)). We provide the models as executable resources for FLORA-2 in [Appendix A](#) of this document.

### 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 a specific use case for booking international train tickets online. We describe a hypothetical Web Service, combining the functionality provided by the Austrian national train operator (ÖBB) and the German Railways (DB), which offer end-user services for searching and buying train tickets for itineraries in Austria and in Germany. This Web Service is composed out of other Web Services, namely one for searching existing train connections, and one for purchasing train tickets online. The services of ÖBB and DB are currently not provided in a machine-processable manner but only via simple end-user web interfaces - Figure 6 shows the currently available portal for ÖBB where the user has to manually select the desired train connection and is lead through the purchase process.

As a user request we assume that the user wants to purchase an international train ticket. The course of the use case shall be the following:

- the customer creates a goal for an international train connection from Innsbruck to Frankfurt on 17th 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 an online ticket per email to the Customer.

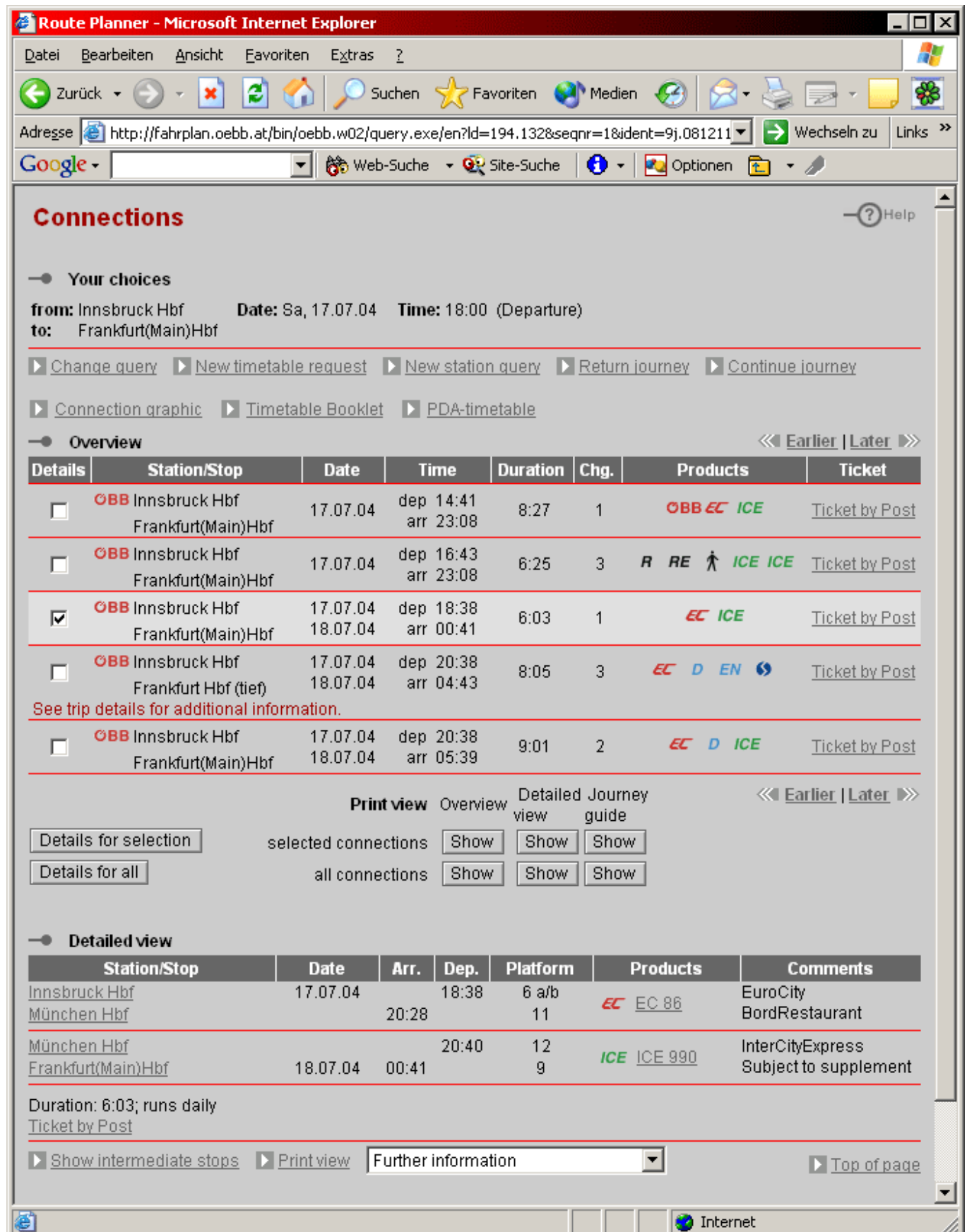


Figure 6. ÖBB Train Connection Itinerary Service

The rationale for choosing this first use case is that it allows to showcase and test describing all WSMO components identified in WSMO Standard. The setting is kept simple on purpose. More complex cases to be added at a later stage of this deliverable can build upon this, providing models of more involved scenarios.

### 3.1.1 Use Case Overview

The following properties have to be covered in our use case modeling. For each of the WSMO top-level components, a separate table describes informally which parts of the use case are concerned.

O1	Ontological information are needed on international train itineraries, on notions of date and time, on the purchasing process, as well as on persons, locations, and addresses. This information should be kept in separate re-usable ontologies, following the modularity principle of ontology design.
O2	An itinerary is described by its start and end locations, date and time of departure and arrival, stations which the train passes (particularly, the station where the border is crossed) and is done by some passenger.
O3	An itinerary describes a valid international train connection.
O4	There has to be traveller / customer that does the itinerar/buys a train ticket
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 and has a price
O7	A ticket is valid for exactly 1 customer
O9	The purchase ontology has to identify the buyer and seller roles, a product with a price, and valid payment methods
O10	We need to be able to express valid payment methods. The only valid payment method for online tickets is credit card payment
O11	Information on Date and Time should allow axiomatic expressions on dependencies of specific dates and times , i.e. expression that define relationships like 'after' or 'before '

G1	Booking an Online Train Ticket
G1.1	From Innsbruck to Frankfurt
G1.2	Start time: 17th July 2004, at 18.00 local time.

W1	A National Train Operator, here the Austrian ÖBB, provides an end-user Web Service that offers a search facility for international train connections and a facility for buying international train tickets online.
W2	The search facility takes a start location, an end location, and a departure date as input and returns a set of itineraries.
W3	The facility for buying train tickets online takes a specific itinerary with start location and end location in countries of its coverage, the information of the customer and the number of his (not expired) credit card number as input, and it returns a ticket for this itinerary as the result.
W4	The user interacts with the end-user Web Service which aggregates the search and purchasing Web Services from possibly different providers like ÖBB, DB, etc.

M1	There need to be OO Mediators that integrate the distinct ontologies used as terminology definitions.
M2	If there are terminological mismatches between the ontologies used in the Goal or the Web Service description, OO Mediators have to be defined to resolve these.
M3	If there are differences between the Goal and the ÖBB-Web Service, a WG Mediator is needed to resolve these.
M4	if there are mismatches between the search facility Web Service and purchase Web Service (which are composed into the end-user Web Service), then a WW Mediator has to be defined which resolves the mismatches.

### 3.1.2 WSMO Modeling

The following provides the modeling of the use case in WSMO with respect to the requirements determined above. The modeling in this document relies on the Web Service Modeling Ontology WSMO, Version 0.2 [Roman et al., 2004]. Some elements of WSMO are not completely specified at the current version of the ontology. This version of the use case modeling is restricted to the WSMO components wherefore a stable specification is existing at this point in time.

#### Ontologies

With regard to modularized ontologies as a basic design principle of WSMO, we define four separate domain ontologies as the terminology definitions for the use case:

1. "International Train Ticket" describes the domain of train tickets
2. "Date and Time" defines a general model for specifying time and dates and relationships of them

3. "Purchase" describes generic elements of purchasing a product between a buyer and a seller.
4. "Locations" describes locations (such as continents, countries and cities and their interrelation).

The ontologies specified in the following are intended to be "real ontologies" in the sense that they describe the specific domain as a shared conceptualization in a sufficient manner. This allows to reuse these ontologies in different settings and use cases - for example, notions of date and time or a general purchase ontology are needed in a lot of other possible scenarios. However, we do not claim the defined below to be such generic ontologies, but they will be enhanced and completed within cooperations with other use cases, projects, and initiatives.

As extensions to WSMO Standard as defined in [[Roman et al., 2004](#)], we apply the following conventions in the Listings:

- **Namespace** With this modeling element we assign a namespace to a prefix, such that this prefix can be used in the subsequent document as shortcut for its representing URI. A URI without namespace prefix in the namespace declaration is used as default prefix for subsequent identifiers (when referencing existing concepts) and the URI with the namespace prefix target-namespace is used as prefix for subsequent identifiers that define new identifiers [[Bray et al. \(Eds.\), 1999](#)], [[Thompson et al. \(Eds.\), 2001](#)]. Note that a namespace declaration does not make any statement about the semantic relation to whatever content is present at that namespace. It is merely used as a unique reference to a particular concept of discourse.
- **Qualified Names** An identifier in WSMML can be: a full URI, a qualified name or a WSMML keyword. WSMML keywords are implicitly preceded by the namespace defined by the prefix "wsml" and highlighted bold in the subsequent listings. Qualified Names are resolved by replacing the prefix with the according URI defined in the namespace declaration. Qualified Names without a Prefix are preceded by the default namespace of the document.
- **Function symbols** A function symbol is a special relation, with a unary range and a n-ary domain. A function symbol with a zero arity domain is a constant. In the following ontologies, e.g. "\*" is a binary function symbol, that takes two numbers as domain and returns one in its range. Note that binary function symbols may be used in infix instead prefix notation to increase readability.
- **Literal vs. Resource** Every element in wsmo is either a Resource or a Literal. Resources are identified by URIs, Literals are enclosed by quotes (""). A Literal can be typed to a data type (e.g. to xsd:integer). Formally a data type is defined by: [[Hayes \(Ed.\), 2004](#)]
  - a non-empty set of character strings called the lexical space of d; e.g. {"true", "1", "false", "0"}
  - a non-empty set called the value space of d; e.g. {true, false}
  - a mapping from the lexical space of d to the value space of d, called the lexical-to-value mapping of d. e.g. {"true", "1"}->{true}; {"false", "0"}->{false}

Furthermore the data type may introduce facets on its value space, such as ordering (and therefore define the axiomatization for the relations "<" and ">"). The XML Schema Definition [Biron & Malhotra, 2001] does this in an informal way. Particular for the data types `xsd:integer`, `xsd:double`, `xsd:float` we tacitly assume the predicates  $X < Y$ ,  $X > Y$ ,  $X \leq Y$ ,  $X \geq Y$ ,  $X = Y$ ,  $X \neq Y$  with the usual meaning "less-than", "greater-then", "less-or-equal", "greater-or-equal", "equal", and "unequal" are defined accordingly to their obvious meaning. Furthermore in these predicates  $X$  and  $Y$  can be arbitrary numeric terms build from members of the value space of the respective `xsd` data types and the binary function symbols "+", "-", "\*", "/", "modulo" and the unary function symbols "-", "floor" and "ceil" with the usual meaning, i.e.

- $X$  is a numeric term if  $X$  is either a typed variable or a literal in the lexical space of `xsd:integer`, `xsd:float`, or `xsd:double`, e.g. "24"<sup>xsd:integer</sup>,  $X$ <sup>xsd:float</sup>, or "1.23"<sup>xsd:double</sup>
- If  $X, Y$  are numeric terms, then also ' $X+Y$ ', ' $X-Y$ ', ' $X*Y$ ', ' $X/Y$ ', ' $-X$ ', ' $(X)$ ', `modulo(X)`, '`floor(X)`', and '`ceil(X)`' are numeric terms, e.g. "12"<sup>xsd:integer</sup> + "24.4"<sup>xsd:float</sup>
- there are no other numeric terms

For brevity, in the listings we allow an abbreviated notation, such as '`12+24.4`' when the respective `xsd` data type is clear from the context. I.e., a number  $x$  without decimal point is by default assumed to stand short for " $x$ "<sup>xsd:integer</sup> and a number with decimal point  $x.y$  is by default assumed to stand short for " $x$ "<sup>xsd:double</sup>.

Currently we are using `xsd:integer`, `xsd:float` and `xsd:string`.

- **Variable Declarations** Variables are declared and typed to a concept, this typing may get refined (by typing to a sub concept) but not extended in the subsequent definitions.
- **Top Concept** For convenience we define a top concept "topConcept" as super concept of all other concept, i.e. every instance is a member of this concept.
- **usedMediators** Note that if the URI of a WSMML ontology is specified as a used mediator, this is a shortcut for importing the respective ontology. This shortcut is not defined by the original definition of the usedMediator attribute in WSMO: In fact, in the following we use this notation as a shortcut for an OOMediator which does neither alter semantics nor requires syntax conversion but simply imports all components of the respective ontology (concepts, relations, instances, etc.) without further modification.
- **Anonymous Ids** For referring to objects, where the identifier is not important, we use anonymous IDs (`_#`, `_#<number>`), as proposed in [Yang and Kifer, 2003]. This is similar to blank nodes in RDF, however there are some differences as outlined in before mentioned paper.

The "International Train Ticket" Ontology defines a train trip and the surrounding concepts as defined the WSMML definition of the ontology shown in Listing 1.

Listing 1. Domain Ontology "International Train Ticket"

**ontology** `http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/tc.wsmml`

**namespace**

`default=http://www.wsmo.org/ontologies/trainConnection#`,

dc=<http://purl.org/dc/elements/1.1#>,  
 wsmi=<http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418#>,  
 dt=<http://www.wsmo.org/ontologies/dateTime#>,  
 prs=<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlPersonMediator.wsmi>,  
 loc=<http://www.wsmo.org/ontologies/location#>,  
 xsd=<http://www.w3.org/2001/XMLSchema#>

**non-functional-properties**

**dc:title** "International Train Connections Ontology"  
**dc:creator** "DERI International"  
**dc:subject** "Train", "Itinerary", "Train Connection", "Ticket"  
**dc:description** "International Train Itineraries"  
**dc:publisher** "DERI International"  
**dc:contributor** "Michael Stollberg", "Ruben Lara", "Holger Lausen", "Axel Polleres"  
**dc:date** "2004-06-28"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/#ontos>  
**dc:format** "text/plain"  
**dc:language** "en-US"  
**dc:relation**  
<http://www.daml.org/2001/06/itinerary/itinerary-ont>,  
<http://daml.umbc.edu/ontologies/ittalks/person>,  
<http://www.wsmo.org/ontologies/dateTime>,  
<http://www.wsmo.org/ontologies/location>,  
<http://opencyc.sourceforge.net/daml/cyc-transportation.daml>  
**dc:coverage** "ID:7029392 Name:World"  
**dc:rights** <http://www.deri.org/privacy.html>  
**version** "\$Revision: 1.7 \$"

**usedMediators****ooMediator**

<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlPersonMediator.wsmi>,  
<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlFactBookMediator.wsmi>,  
<http://www.wsmo.org/ontologies/dateTime>,  
<http://www.wsmo.org/ontologies/location>

**comment:** conceptDefinitions

**concept** station **subconceptOf** loc:location

**non-functional-properties**

**dc:description** "Train station"

code **oftype** xsd:string

**non-functional-properties**

**dc:description** "Code of the station"

locatedIn **oftype set** loc:location

borderToCountry **oftype** loc:location

**non-functional-properties**

**dc:description** "For stations located at the border"

**concept** itinerary

**non-functional-properties**

**dc:description** "An itinerary between two locations"

passenger **oftype** prs:person

**non-functional-properties**

**dc:description** "prs:person is a subset of vCard (<http://www.ietf.org/rfc/rfc2425.txt>)"

recordLocatorNumber **oftype** xsd:string

trip **oftype** trip

**concept** trip

start **oftype** loc:location

end **oftype** loc:location

via **oftype set** loc:location

departure **oftype** dt:dateAndTime

arrival **oftype** dt:dateAndTime

duration **oftype** dt:interval

distance **oftype** loc:distance

**concept** trainTrip **subconceptOf** trip

**non-functional-properties**

**dc:description** "A train trip"

start **oftype** station

end **oftype** station

via **oftype set** station

seat **oftype** xsd:string

train **oftype** xsd:string

class **oftype** xsd:string

**comment:** variableDefinitions

**variable** S **memberOf** station  
**variable** L, Start, End **memberOf** loc:location  
**variable** C **memberOf** loc:country  
**variable** T **memberOf** trip  
**variable** D, A **memberOf** dt:dateAndTime

**comment:** axiomDefinitions

**axiom** stationCountry  
**non-functional-properties**  
**dc:description** "Integrity constraint: if a station is located in a place which is located in a given country, the country of the station is the same"  
**logical-expression**  
 "<-  
 S[  
   locatedIn **hasvalue** L,  
   country **hasvalue** C]  
 and not L[  
   country **hasvalue** C]."

**axiom** departureBeforeArrival  
**non-functional-properties**  
**dc:description** "Integrity Constraint: departure has to be before arrival"  
**logical-expression**  
 "<-  
 T[  
   departure **hasvalue** D,  
   arrival **hasvalue** A]  
 and A <= D."

**axiom** startNotEqualEnd  
**non-functional-properties**  
**dc:description** "Integrity Constraint: the start and end of a trip have to be different"  
**logical-expression**  
 "<-  
 T[  
   start **hasvalue** Start,  
   end **hasvalue** End]  
 and Start = End."

**comment:** instanceDefinitions

**comment:** A link to large set of instances is missing in WSMO.  
 Therefore, in this version of the ontology we only include some example instances. The inclusion of links to large set of instances will be considered in future versions of WSMO

**instance** innsbruckHbf **memberOf** station  
 name **hasvalue** "Innsbruck Hbf"  
 code **hasvalue** "INN"  
 locatedIn **hasvalues** {loc:innsbruck}

**instance** frankfurtHbf **memberOf** station  
 name **hasvalue** "Frankfurt Hbf"  
 code **hasvalue** "FKF"  
 locatedIn **hasvalues** {loc:frankfurt}

Please notice that the link to large set of instances is missing in WSMO. Therefore, in this version of the ontology we only include some example instances, which holds for the other ontologies defined in this use case as well. The inclusion of links to large set of instances will be considered in future versions of WSMO.

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.

The main ontology taken into consideration for developing this conceptual model of Date and Time is an entry sub-ontology of time, available at <http://www.isi.edu/~pan/damlltime/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 calendarclock ontology (<http://daml.umbc.edu/ontologies/cobra/0.4/calendarclock>) 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/>), which is based on ISO 8601.

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

```
ontology http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/dt.wsml

namespace
  default=http://www.wsmo.org/ontologies/dateTime#,
  dc=http://purl.org/dc/elements/1.1#,
  wsmi=http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418#

non-functional-properties
  dc:title "Date and Time Ontology"
  dc:creator "DERI International"
  dc:subject "Date", "Time", "Date and Time Algebra"
  dc:description "generic representation of data and time including basic algebra"
  dc:publisher "DERI International"
  dc:contributor "Holger Lausen", "Axel Polleres", "Ruben Lara"
  dc:date "2004-06-28"
  dc:type http://www.wsmo.org/2004/d2/v0.3/20040329/#ontos
  dc:format "text/plain"
  dc:language "en-US"
  dc:relation http://www.isi.edu/~pan/damlltime/time-entry.owl,
  http://www.w3.org/TR/xmlschema-2/
  dc:coverage "World"
  dc:rights http://www.deri.org/privacy.html
  version "$Revision: 1.12 $"

comment: conceptDefinitions
concept instant
  non-functional-properties
    dc:description "An instant represents a particular point in time and is the superconcept
    of all concrete representations such as the Gregorian calendar"

concept interval
  non-functional-properties
    dc:description "An interval represents a duration between 2 points in time"
    start otype instant
    end otype instant

concept date
  non-functional-properties
    dc:description "concept date and its representation according to the Gregorian Calendar"
    subconcept-of instant
    dayOfMonth otype dayOfMonth
    monthOfYear otype monthOfYear
```

year **of**type year

**concept** dayOfMonth

**non-functional-properties**

**dc:description** "day of a month is represented by an integer"  
subconcept-of xsd:integer

**concept** year

**non-functional-properties**

**dc:description** "year is represented by an integer"  
subconcept-of xsd:integer

**concept** monthOfYear

**non-functional-properties**

**dc:description** "monthOfYear is represented by an integer"  
subconcept-of xsd:integer

**concept** time

hourOfDay **of**type hourOfDay

minuteOfHour **of**type minuteOfHour

secondOfMinute **of**type secondOfMinute

**concept** secondOfMinute

**non-functional-properties**

**dc:description** "a secondOfMinute is represented by an integer"  
subconcept-of xsd:integer

**concept** minuteOfHour

**non-functional-properties**

**dc:description** "a minuteOfHour is represented by an integer"  
subconcept-of xsd:integer

**concept** hourOfDay

**non-functional-properties**

**dc:description** "a hourOfDay is represented by an integer"  
subconcept-of xsd:integer

**concept** dateAndTime

**non-functional-properties**

**dc:description** "concept date and time and representing together a specific point of time (instant)"  
subconcept-of instant

date **of**type date

time **of**type time

**comment:** variableDefinitions

**variable** X, Y, Z, D1, D2 **memberOf** topConcept

**variable** A, B, C, D, E, F, JDN, JDN\_D1, JDN\_D2, SFM\_T1, SFM\_T2 **memberOf** xsd:integer

**variable** T, T1, T2 **memberOf** time

**comment:** functionDefintions

**function** julianDayNumber

**non-functional-properties**

**dc:description** "The Julian Day Count is a uniform count of days from a remote epoch in the past (about 4712 BC). At this instant, the Julian Day Number is 0. Once you have the Julian Day Number of a particular date in history, it is easy to calculate time elapsed between it and any other Julian Day Number"

**dc:source** <http://quasar.as.utexas.edu/BillInfo/JulianDatesG.html>

**parameter** instant **of**type instant

**non-functional-properties**

**dc:description** "For each instant there should exist a corresponding Julian Day Number, however it may not be always defined only by this binary predicate, e.g. if the instant is represented as Gregorian Date and it is a date between 1582 and 1924. A country must be given as third parameter (since e.g. Greece changed no earlier then 9th of March 1924 from the Julian to the Gregorian Calendar)"

**comment:** The following dc:source indicates which country changed in which year

**comment:** from the Julian to the Gregorian Calendar

**dc:source** <http://members.brabant.chello.nl/~h.reints/cal/whenjul2greg.htm>

**range** **of**type xsd:integer

**function** daysBetween

**non-functional-properties**

**dc:description** "(Instant1, Instant2, Difference) is a triple of the ternary relation corresponding to this function iff Instant1 and Instant2 are members of the concept 'instant' (particular point in time) and Instant2 is 'Difference' days after Instant1."

**parameter** instant1 **of**type instant

**parameter** instant2 **oftype** instant  
**range oftype** xsd:integer

**function** secondsBetween

**non-functional-properties**

**dc:description** "(Instant1, Instant2, Difference) is a triple of the ternary relation corresponding to this function iff Instant1 and Instant2 are members of the concept 'instant' (particular point in time) and Instant2 is 'Difference' seconds after Instant1."

**parameter** instant1 **oftype** instant

**parameter** instant2 **oftype** instant

**range oftype** xsd:integer

**function** secondsFromMidnight

**non-functional-properties**

**dc:description** "(Time, SecondsFromMidnight) is a tuple of the binary relation corresponding to this function iff SecondsFromMidnight are the seconds elapsed from 00:00:00 of the same day.

This simplifies the axiomatization of the difference between two given times"

**parameter** time **oftype** time

**range oftype** xsd:integer

**comment:** relationDefintions

**relation** contains

**non-functional-properties**

**dc:description** "(Interval, X) is a tuple of the binary relation corresponding to this function iff Interval contains X and X is an instant or an interval"

**parameter** interval **oftype** interval

**parameter** intervalOrInstant **oftype** (instant or interval)

**comment:** axiomDefinitions

**axiom** invalidMonthOfYear

**non-functional-properties**

**dc:description** "integrity constraint for valid monthOfYear"

**logical-expression**

"<-

X memberOf monthOfYear and  
(X < 1 or X > 12)."

**axiom** invalidDayOfMonth

**non-functional-properties**

**dc:description** "integrity constraint for valid dayOfMonths"

**logical-expression**

"<-

X memberOf dayOfMonth and  
(X < 1 or X > 31)."

**axiom** validDate

**non-functional-properties**

**dc:description** "Integrity Constraints for date.

The dayOfMonth is valid in dependency of the actual monthOfYear, in a leap year the month 2 of the Year has 29 days otherwise 28. For leap years holds the following: Every year divisible by 4 is a leap year.

However, every year divisible by 100 is not a leap year; and every year divisible by 400 is a leap year after all.

Note: This axiomatization is still imprecise, since the country plays a role when defining a valid day of the month: E.g. 1712 was a double leap year in Sweden, i.e. February 1712 had 30 days in Sweden.

The mathematical function symbol modulo is assumed to be defined elsewhere as that it returns the remainder after an integer division of its first argument by its second"

**dc:source** <http://www.tondering.dk/claus/cal/node3.html>

**logical-expression**

"<-

X memberOf date and (  
(X.dayOfMonth > 28 and X.monthOfYear = 2,  
not ((modulo(X.year ,4) = 0 and not modulo(X.year ,100) = 0)  
or modulo(X.year ,400) = 0))  
or (X.dayOfMonth > 29 , X.monthOfYear = 2)  
or (X.dayOfMonth > 30 , X.monthOfYear = 4)  
or (X.dayOfMonth > 30 , X.monthOfYear = 6)  
or (X.dayOfMonth > 30 , X.monthOfYear = 9)

or (X.dayOfMonth > 30 , X.monthOfYear = 11)  
)."

**axiom** invalidHourOfDay

**non-functional-properties**

**dc:description** "integrity constraint for valid hourOfDay:"

**logical-expression**

"<-

X **memberOf** hourOfDay **and**  
(X < 0 **or** X >= 24)."

**axiom** invalidMinuteOfHour

**non-functional-properties**

**dc:description** "integrity constraint for valid minuteOfHour:"

**logical-expression**

"<-

X **memberOf** minuteOfHour **and**  
(X < 0 **or** X >= 60)."

**axiom** invalidSecondOfMinute

**non-functional-properties**

**dc:description** "integrity constraint for valid secondOfMinute:"

**logical-expression**

"<-

X **memberOf** secondOfMinute **and**  
(X < 0 **or** X >= 60)."

**axiom** invalidInterval

**non-functional-properties**

**dc:description** "computes if a interval X contains a second interval Y"

**logical-expression**

"<-

X **memberOf** interval **and** X.start >= X.end."

**axiom** equalityDate

**non-functional-properties**

**dc:description** "computes equality of a date"

**logical-expression**

"X = Y <-

Y **memberOf** date **and** X **memberOf** date **and**  
X.dayOfMonth = Y.dayOfMonth **and**  
X.monthOfYear = Y.monthOfYear **and**  
X.year = Y.year."

**axiom** beforeDate

**non-functional-properties**

**dc:description** "computes if a given date X is before another date Y"

**logical-expression**

"X < Y <-

Y **memberOf** date **and** X **memberOf** date **and**  
((X.dayOfMonth = Y.dayOfMonth **and** X.monthOfYear = Y.monthOfYear **and** X.year = Y.year) **or**  
(X.monthOfYear < Y.monthOfYear **and** X.year = Y.year) **or**  
(X.year < Y.year))."

**axiom** afterDate

**non-functional-properties**

**dc:description** "defined as inverse of beforeDate"

**logical-expression**

"X > Y <- Y < X"

**axiom** julianDayNumber

**non-functional-properties**

**dc:description** "This axiom describes how the correct Julian Day Number

can be computed for a given Gregorian Calendar Date. Note  
that the Gregorian Calendar was introduced in 15.October 1582.  
however until 1919 this axiomatization is not unambiguous since the country  
should be taken into to account as 3rd parameter (e.g. Greece  
changed at the 9 Mar 1924 from the Julian to the Gregorian calendar).

Details to the axiomatization

If the month is January or February we subtract 1 from the year to get a new Year  
and add 12 to the month to get a new Month. (Thus, we are thinking of January and  
February as being the 13th and 14th month of the previous year and March is the  
start of the year, this simplifies the calculation considering the leap year)

Within the calculation the fractional part of all results has to be dropped, here we use the function symbol floor() [it can be rewritten as predicate, however it gets less readable]

A more lengthy description of this axiomatization can be found at <http://quasar.as.utexas.edu/BillInfo/JulianDatesG.html>

**dc:source** <http://quasar.as.utexas.edu/BillInfo/JulianDatesG.html>,  
<http://members.brabant.chello.nl/~h.reints/cal/whenjul2greg.htm>

**logical-expression**

```
"julianDayNumber(X) = JDN <-
  X memberOf date and
  ((
    X.monthOfYear < 3 and
    Y = X.year - 1 and
    M = X.monthOfYear + 12
  )
  or
  (
    X.monthOfYear > 2 and
    Y = X.year and
    M = X.monthOfYear
  ))
  and
  D = X.dayOfMonth and
  A = floor(Y / 100) and
  B = floor(A / 4) and
  C = 2 - A + B and
  E = floor(365.25 * (Y + 4716)) and
  F = floor(30.6001 * (M + 1)) and
  JDN = C + D + E + F - 1524."
```

**axiom** daysBetweenDates

**non-functional-properties**

**dc:description** "the difference in days between 2 dates"

**logical-expression**

```
"daysBetween(D1, D2) hasvalue X <-
  D1 memberOf date and D2 memberOf date and
  X = julianDayNumber(D1) - julianDayNumber(D2)."
```

**axiom** equalityTime

**non-functional-properties**

**dc:description** "computes if two given times are the same"

**logical-expression**

```
"X = Y <-
  X memberOf time and Y memberOf time and
  X.secondOfMinute = Y.secondOfMinute and
  X.minuteOfHour = Y.minuteOfHour and
  X.hourOfDay = Y.hourOfDay."
```

**axiom** beforeTime

**non-functional-properties**

**dc:description** "computes if a given time X is before another time Y"

**logical-expression**

```
"X < Y <-
  X memberOf time and Y memberOf time and
  ((X.secondOfMinute < Y.secondOfMinute and X.minuteOfHour = Y.minuteOfHour and X.hourOfDay =
  Y.hourOfDay) or
  (X.minuteOfHour < Y.minuteOfHour and X.hourOfDay = Y.hourOfDay) or
  (X.hourOfDay < Y.hourOfDay))."
```

**axiom** afterTime

**non-functional-properties**

**dc:description** "defined as inverse of beforeTime"

**logical-expression**

```
"X > Y <- Y < X."
```

**axiom** secondsFromMidnight

**non-functional-properties**

**dc:description** "computes the amount of seconds from midnight"

**logical-expression**

```
"secondsFromMidnight(T) hasvalue X <-
  T memberOf time and
  X = T.secondOfMinute + (T.minuteOfHour*60) + (T.hourOfDay*60*60)."
```

**axiom** secondsBetweenTimes

```

non-functional-properties
  dc:description "the difference in seconds between 2 times"
logical-expression
  "secondsBetween(T1, T2) hasvalue X <-
    T1 memberOf time and T2 memberOf time and
    X = secondsFromMidnight(T1) - secondsFromMidnight(T2)."
```

**axiom** equalityDateAndTime

```

non-functional-properties
  dc:description "computes if Date and Time are equal"
logical-expression
  "X = Y <-
    X memberOf dateAndTime and Y memberOf dateAndTime and
    X.date = Y.date and
    X.time = Y.time."
```

**axiom** beforeDateAndTime

```

non-functional-properties
  dc:description "computes if a given date and time X is before another date and time Y"
logical-expression
  "X < Y <-
    X memberOf dateAndTime and Y memberOf dateAndTime and
    ((X.date = Y.date and X.time < Y.time) or
    X.date < Y.date)."
```

**axiom** afterDateAndTime

```

non-functional-properties
  dc:description "defined as inverse of beforeDateAndTime"
logical-expression
  "X > Y <- X
    memberOf dateAndTime and Y memberOf dateAndTime and
    Y < X."
```

**axiom** secondsBetweenDateAndTime

```

non-functional-properties
  dc:description "computes the difference in seconds between two different DateAndTime"
logical-expression
  "secondsBetween(D1, D2) = X <-
    D1 memberOf dateAndTime and D2 memberOf dateAndTime and
    X = secondsFromMidnight(D1.time) + julianDayNumber(D1.date) * 24 * 60 * 60 -
    (secondsFromMidnight(D2.time) + julianDayNumber(D2.date) * 24 * 60 * 60)."
```

**axiom** daysBetweenDateAndTime

```

non-functional-properties
  dc:description "the difference in days between two different DateAndTime"
logical-expression
  "daysBetween(D1, D2) hasValue X <-
    D1 memberOf dateAndTime and D2 memberOf dateAndTime and
    X = daysBetween(D1.date, D2.date)."
```

**axiom** intervalContainment

```

non-functional-properties
  dc:description "computes if a interval X contains a second interval Y"
logical-expression
  "contains(X, Y) <-
    X memberOf interval and Y memberOf interval and
    (X.start < Y.start or X.start = Y.start) and
    (X.end > Y.end or X.end = Y.end)."
```

**axiom** instantContainment

```

non-functional-properties
  dc:description "computes if a interval X contains a instant Y"
logical-expression
  "contains(X, Y) <-
    X memberOf interval and Y memberOf instant and
    (X.start < Y or X.start = Y) and
    (X.end > Y or X.end = Y)."
```

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). At the current state this is a preliminary domain ontology for products, purchases, and payment methods. This ontology will be re-designed and enhanced in future

versions. In order to provide a general purchase ontology, several existing conceptualizations are considered as a starting point, inter alia:

- IOTP ( The Internet Open Trading Protocol), website: <http://www.ietf.org/html.charters/trade-charter.html>  
this framework seems to be suitable for our needs in principal, although there does not seem to exist stable OWL or similar ontology formulations of these ontologies. IOTP also has an additional supplement for modeling payment methods)
- ebXML, website: <http://www.ebxml.org>  
is another effort for aligning standards in the modeling of business processes, but we did not find an ontology formulation of this standards either so far.
- Another possibility is to develop an ontology based on EDIFACT [[EDIFACT](#)] in the course of WSMO.

### Listing 3. Domain Ontology "Purchase"

```
ontology http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/po.wsmml

namespace
  default=http://www.wsmo.org/ontologies/purchase#,
  dc=http://purl.org/dc/elements/1.1#,
  wsmml=http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418#,
  loc=http://www.wsmo.org/ontologies/location#,
  cu=http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlCurrencyMediator.wsmml#,
  dt=http://www.wsmo.org/ontologies/dateTime#

non-functional-properties
  dc:title "Purchase Ontology"
  dc:creator "DERI International"
  dc:subject "Buyer", "Seller", "Product", "Price",
    "Payment method", "Delivery"
  dc:description "general purchase ontology"
  dc:publisher "DERI International"
  dc:contributor "Michael Stollberg", "Axel Polleres", "Ruben Iara", "Holger Lausen"
  dc:date "2004-06-28"
  dc:type http://www.wsmo.org/2004/d2/v0.3/20040329/#ontos
  dc:format "text/plain"
  dc:language "en-US"
  dc:relation
    http://www.daml.ecs.soton.ac.uk/ont/currency.daml,
    http://www.wsmo.org/ontologies/location,
    http://www.wsmo.org/ontologies/dateTime
  dc:coverage "ID:7029392 Name:World"
  dc:rights http://www.deri.org/privacy.html
  version "$Revision: 1.16 $"

usedMediators
  ooMediators
    http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlCurrencyMediator.wsmml,
    http://www.wsmo.org/ontologies/location,
    http://www.wsmo.org/ontologies/dateTime

comment: conceptDefinitions

concept buyer
  non-functional-properties
    dc:description "Buyer of some items"
  shipTo oftype loc:address
  billTo oftype loc:address
  purchaseIntention oftype set tradeltem
  hasPayment oftype set paymentMethod

concept seller
  non-functional-properties
    dc:description "Seller of some items"
  address oftype loc:address
  saleIntention oftype set tradeltem
  acceptsPayment oftype set paymentMethod
```

```

concept tradeItem
  non-functional-properties
    dc:description "A trade item"
  product oftype product
  pricelimit oftype price

concept product
  non-functional-properties
    dc:description "Generic product"
  name oftype xsd:string

concept price
  non-functional-properties
    dc:description "Generic price"
  amount oftype xsd:float
  currency oftype cu:currency

concept paymentMethod
  non-functional-properties
    dc:description "Payment method for a trade"
  name oftype xsd:string

concept trade
  non-functional-properties
    dc:description "A trade is an actual agreement on trading items between two trading partners"
  items oftype set tradeItem
  buyer oftype buyer
  seller oftype seller
  payment oftype paymentMethod

concept delivery
  non-functional-properties
    dc:description "Delivery of a good as an effect of a purchase"
  products oftype set product
  receiver oftype buyer
  sender oftype seller

concept creditCard subconceptOf paymentMethod
  non-functional-properties
    dc:description "A credit card"
  holder oftype xsd:string
  expMonth oftype dt:monthOfYear
  expYear oftype dt:year
  type oftype xsd:string

concept cash subconceptOf paymentMethod
  non-functional-properties
    dc:description "Cash payment method"
  currency oftype cu:currency

comment: variableDefinitions

  variable T memberOf tradeItem
  variable P memberOf price

comment: instanceDefinitions

comment: A link to large set of instances is missing in WSMO.
  The inclusion of links to large set of instances will be considered
  in future versions of WSMO.
  A complete list of currencies can be found at
  http://www.daml.ecs.soton.ac.uk/ont/currency.daml

```

The "Locations Ontology" defines an concepts for locations, including cities and states, as well as postal addresses. This general ontology builds upon the OWL-Factbook and Address ontology, and it can be re-used in different settings.

Listing 4. Domain Ontology "Locations"

**ontology** <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/loc.wsml>

**namespace**

default=<http://www.wsmo.org/ontologies/location#>,  
 dc=<http://purl.org/dc/elements/1.1#>,  
 wsmi=<http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418#>,  
 cnt=<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlFactbookMediator.wsmi#>,  
 ad=<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlAddressMediator.wsmi#>,  
 xsd=<http://www.w3.org/2001/XMLSchema#>

**non-functional-properties**

**dc:title** "Locations Ontology"  
**dc:creator** "DERI International"  
**dc:subject** "Location", "Country", "State", "City", "Address"  
**dc:description** "Ontology for representing ontologies in the current political/social system"  
**dc:publisher** "DERI International"  
**dc:contributor** "Ruben Lara", "Axel Polleres"  
**dc:date** "2004-06-28"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/#ontos>  
**dc:format** "text/plain"  
**dc:language** "en-US"  
**dc:relation**  
<http://www.daml.org/2001/09/countries/fips-10-4-ont>,  
<http://www.daml.org/2001/09/countries/iso-3166-ont>,  
<http://www.daml.org/2003/09/factbook/factbook-ont>,  
<http://daml.umbc.edu/ontologies/ittalks/address>  
**dc:coverage** "ID:7029392 Name:World"  
**dc:rights** <http://www.deri.org/privacy.html>  
**version** "\$Revision: 1.7 \$"

**usedMediators****ooMediator**

<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlFactbookMediator.wsmi>,  
<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlAddressMediator.wsmi>

**comment:** conceptDefinitions

**concept** location**non-functional-properties**

**dc:description** "General notion of location"  
 name **of type** xsd:string  
 country **of type set** country

**concept** country **subconceptOf** cnt:country**non-functional-properties**

**dc:description** "Add the codes to the CIA country properties"  
**comment:** FIPS 10-4 Country Code  
 fipsCode **of type** xsd:string  
**comment:** ISO 3166 Country Code  
 isoCode **of type** xsd:string

**concept** address **subconceptOf** ad:address**non-functional-properties**

**dc:description** "Extended address, adding more details to  
 city, state and country"  
 city **of type** city  
 state **of type** state  
 country **of type** country

**concept** city **subconceptOf** location**non-functional-properties**

**dc:description** "City"  
 state **of type** state  
 population **of type** xsd:integer  
 extension **of type** xsd:integer  
**non-functional-properties**  
**dc:description** "Extension of the city in square kilometers"  
 zipcodes **of type set** xsd:string

**concept** state **subconceptOf** location**non-functional-properties**

**dc:description** "State"  
 cities **of type set** city  
 population **of type** xsd:integer  
 extension **of type** xsd:integer

**concept** border **subconceptOf** location**non-functional-properties**

**dc:description** "Border between two countries. Notice that it would be more natural to model this as a location with a cardinality constraint = 2 for the country property. However, it is not clear how to do this in F-Logic"  
**countryA otype** country  
**countryB otype** country

**concept** distance  
**non-functional-properties**  
**dc:description** "Distance between two points"  
**amount otype** xsd:float  
**units otype** xsd:string  
**kilometers otype** xsd:float  
**miles otype** xsd:float

**comment:** variableDefinitions

**variable** D, D1, D2 **memberOf** distance  
**variable** U **memberOf** xsd:string  
**variable** A **memberOf** xsd:float

**comment:** axiomDefinitions

**axiom** validDistance  
**non-functional-properties**  
**dc:description** "The amount in a distance cannot be less than 0. We only accept kilometers and miles."  
**logical-expression**  
 "<-  
 D[  
   **amount hasvalue** A,  
   **units hasvalue** U]  
**and** A < 0  
**and not** (U="Kilometers" **or** U="Miles")."

**axiom** kilometers  
**non-functional-properties**  
**dc:description** "Calculation for the kilometers property of distance"  
**logical-expression**  
 "D[  
   **amount hasvalue** A,  
   **units hasvalue** U] **and**  
 ((U="Kilometers" **and** kilometers=A) **or**  
 (U="Miles" **and** kilometers=A\*1.609344))."

**axiom** miles  
**non-functional-properties**  
**dc:description** "Calculation for the miles property of distance"  
**logical-expression**  
 "D **memberOf** distance[  
   **amount hasvalue** A,  
   **units hasvalue** U] **and**  
 ((U="Miles" **and** miles=A) **or**  
 (U="Kilometers" **and** miles=A/1.609344))."

**axiom** equalityDistance  
**non-functional-properties**  
**dc:description** "Computes equality of a distance"  
**logical-expression**  
 "D1 = D2 <-  
 D1.kilometers = D2.kilometers."

**axiom** lessThanDistance  
**non-functional-properties**  
**dc:description** "Computes -less than- for a distance"  
**logical-expression**  
 "D1 < D2 <-  
 D1.kilometers < D2.kilometers."

**axiom** moreThanDistance  
**non-functional-properties**  
**dc:description** "Computes -more than- for a distance"  
**logical-expression**  
 "D1 > D2 <-  
 D1.kilometers > D2.kilometers."

**comment:** instanceDefinitions  
**comment:** "A link to large set of instances is missing in WSMO. Therefore, in this version of the ontology we only include some example instances. The inclusion of links to large set of instances will be considered in future versions of WSMO"

**instance** austria **memberOf** country  
 fipsCode **hasvalue** "AU"  
 isoCode **hasvalue** "AT"

**instance** germany **memberOf** country  
 fipsCode **hasvalue** "GM"  
 isoCode **hasvalue** "DE"

**instance** usa **memberOf** country  
 fipsCode **hasvalue** "US"  
 isoCode **hasvalue** "US"

**instance** innsbruck **memberOf** city  
 name **hasvalue** "Innsbruck"  
 country **hasvalue** austria

**instance** frankfurt **memberOf** city  
 name **hasvalue** "Frankfurt"  
 country **hasvalue** germany

**instance** boston **memberOf** city  
 name **hasvalue** "Boston"  
 country **hasvalue** usa

**instance** massachusetts **memberOf** state  
 name **hasvalue** "Massachusetts"  
 country **hasvalue** usa

## Goals

Goals denote what a user wants as the result of the Web Service. For modeling the goal, we describe the information elements that the user wants to get from the service (the postcondition) together with the state of the world desired after the service execution (the effect).

In our use case, we have one Goal: a user wants to buy a train itinerary from Innsbruck to Frankfurt on a certain date. The Goal states that the desire is to get the description of the itinerary bought, and that the effect of the Web Service has to be a trade between the train company and the requester for the desired itinerary. Listing 5 shows this Goal with the following elements:

- **postcondition:** A itinerary for a train trip from Innsbruck to Frankfurt on July, 17th 2004, valid for the customer Tim Berners-Lee.
- **effect:** there shall be a trade for the itinerary, to be payed with a given creditcard and, the bill of the trade will be sent to Tim's address, and the actual ticket will also be sent to Tim's address.

### Listing 5: Goal - buying a train ticket online

```
goal http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/goal.wsm1

namespace
  default=http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/goal#,
  dc=http://purl.org/dc/elements/1.1#,
  dt=http://www.wsmo.org/ontologies/dateTime#,
  tc=http://www.wsmo.org/ontologies/trainConnection#,
  po=http://www.wsmo.org/ontologies/purchase#,
```

loc=<http://www.wsmo.org/ontologies/location#>,  
 wsml=<http://www.wsmo.org/2004/d2/#>

#### non-functional-properties

**dc:title** "Buying a ticket for a train trip"  
**dc:creator** "DERI International"  
**dc:subject** "Train Tickets", "Online Ticket Booking", "Train trip"  
**dc:description** "Express the goal of buying a ticket for a train trip"  
**dc:publisher** "DERI International"  
**dc:contributor** "Michael Stollberg", "Ruben Lara", "Holger Lausen", "Axel Polleres"  
**dc:date** "2004-06-07"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/#L3958>  
**comment:** MIME type according to [RFC2646,RFC2046]  
**dc:format** "text/plain"  
**comment:** langugae definition according [RFC3066, ISO639]  
**dc:language** "en-us"  
**dc:relation**  
<http://www.wsmo.org/ontologies/dateTime>,  
<http://www.wsmo.org/ontologies/trainConnection>,  
<http://www.wsmo.org/ontologies/purchase>,  
<http://www.wsmo.org/ontologies/location>  
**dc:coverage** "ID:7029392 Name:World"  
**dc:rights** <http://deri.at/privacy.html>  
**version** "\$Revision: 1.8 \$"

#### usedMediators

<http://www.wsmo.org/ontologies/dateTime>,  
<http://www.wsmo.org/ontologies/trainConnection>,  
<http://www.wsmo.org/ontologies/purchase>,  
<http://www.wsmo.org/ontologies/location>

#### postcondition

**axiom** buyATicketForItinerary

##### non-functional-properties

**dc:description** "The goal postcondition is represented as a fact, in this case the fact is only specified partly, e.g. for the time of departure the minute and seconds are not specified.  
 It represents that 'Tim Berners-Lee' wants to go from innsbruckHbf to frankfurtHbf departing from innsbruckHbf at 17.07.2004 18h"

##### logical-expression

```
"someItinerary memberOf tc:itinerary[
  trip hasValue someTrip memberOf tc:trainTrip[
    start hasValue innsbruckHbf,
    end hasValue frankfurtHbf,
    departure hasValue _# memberOf dt:dateAndTime[
      date hasValue _# memberOf dt:date[
        dayOfMonth hasValue 17,
        monthOfYear hasValue 7,
        year hasValue 2004
      ],
      time hasValue _# memberOf dt:time[
        hourOfDay hasValue 18
      ]
    ],
    passenger hasValue _# memberOf loc:person[
      firstName hasValue "Tim",
      lastName hasValue "Berners-Lee",
      email hasValue "timbl@w3.org"
    ]
  ]
]."
```

#### effect

**axiom** havingTradeForTrip

##### non-functional-properties

**dc:description** "The goal effect is represented as a fact  
 It represents that 'Tim Berners-Lee' wants to have a trade with a provider (not specified) for the itinerary given; the ticket should be delivered to his address and he wants to pay by credit card"

##### logical-expression

```
"someTrade memberOf po:trade[
  items hasValues someTrip,
  buyer hasValue _# memberOf po: buyer[
    shipTo hasValue timsAddress memberOf loc:address[
```

```

        roomNumber hasValue 3,
        streetAddress hasValue 'Tims street',
        city hasValue boston,
        state hasValue massachusetts,
        zip hasValue 02103
    ],
    billTo hasValue timsAddress
],
payment hasValue _# memberOf po:creditCard[
    holder hasValue 'Tim Berners-Lee',
    expMonth hasValue 9,
    expYear hasValue 2007,
    type hasValue 'MasterCard'
]
]"

```

Notice that an instance of the concept 'Itinerary' is used as the value of the property 'Items' of the concept 'Trade'. In the ontologies defined above, Itinerary is not defined in tc.wsml as a subconcept of po:product. This subclassing should be done by an OO-mediator that imports the terminology required for the goal and takes care of this operation. Such a mediator will be included in the next version of this deliverable.

## Web Services

As explained above, we define one (imaginary) Web Service in this use case: an end-user service (means that the user interacts with this service) for purchasing international train tickets offered by the Austrian national train operator ÖBB, which is composed of other Web Services, each for the search and buying facility of international train tickets. This setting allows modeling all notions of a WSMO Web Service description: A Capability of the end-user service and its Choreography for user-service interaction, as well as the orchestration which incorporates the aggregated Web Services. The current version of WSMO Standard does only provide a stable specification for describing Capabilities, the model below is restricted to the overall Web Service description and the Capability definition. The modeling for the WSMO Web Service Interface will be added in a later version.

A Web Service Capability in WSMO is described by pre- and postconditions, assumptions and effects, as defined in [Roman et al., 2004]. More detailed discussion of the Discovery mechanism of WSMO Goals and Capabilities is provided in section [3.1.3](#).

- **precondition:** the input has to be the information about the buyer, and the purchase intention of the buyer has to be a train itinerary with start and end locations in Austria or Germany. Furthermore, the departure date for the trip has to be after the current date and the payment method of the buyer has to be a valid (not expired) credit card.
- **postcondition:** describes the possible trips the service can return wherefore the start and end location have to be either in Austria or Germany.
- **effect:** a trade is performed for the train itinerary given as a postcondition.

Listing 6: ÖBB Web Service for Booking Online Train Tickets for Austria and Germany

**webservice** <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040526/resources/ws.wsml>

**namespace**

default=<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040526/resources/ws#>  
 dc=<http://purl.org/dc/elements/1.1#>,  
 wsmo=<http://www.wsmo.org/2004/d2/#>  
 dt=<http://www.wsmo.org/ontologies/dateTime#>,  
 tc=<http://www.wsmo.org/ontologies/trainConnection#>,  
 po=<http://www.wsmo.org/ontologies/purchase#>,  
 loc=<http://www.wsmo.org/ontologies/location#>,

**non-functional-properties**

**dc:title** "ÖBB Online Ticket Booking Web Service"  
**dc:creator** "DERI International"  
**dc:subject**  
**dc:description** "web service for booking online train tickets for Austria and Germany"  
**dc:publisher** "DERI International"  
**dc:contributor** "Michael Stollberg", "Ruben Lara", "Holger Lausen"  
**dc:date** "2004-06-03"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/##L3966>  
**comment:** MIME type according to [RFC2646,RFC2046]  
**dc:format** "text/plain"  
**comment:** language definition according [RFC3066, ISO639]  
**dc:language** "en-us"  
**dc:relation** <http://www.wsmo.org/ontologies/dateTime>,  
<http://www.wsmo.org/ontologies/trainConnection>,  
<http://www.wsmo.org/ontologies/purchase>,  
<http://www.wsmo.org/ontologies/location>  
**dc:coverage** tc:austria, tc:germany  
**dc:rights** <http://deri.at/privacy.html>  
**version** "\$Revision: 1.1 \$"

**usedMediators**

<http://www.wsmo.org/ontologies/dateTime>,  
<http://www.wsmo.org/ontologies/trainConnection>,  
<http://www.wsmo.org/ontologies/purchase>,  
<http://www.wsmo.org/ontologies/location>

**capability**

**preCondition**

**non-functional-properties**

**dc:description** "the input has to be a buyer with a purchase intention for an itinerary wherefore the start- and endlocation have to be in Austria or in Germany, and the departure date has to be later than the current Date. Also, a not-expired credit card for payment is expected."

**logical-expression**

" inputBuyer **memberOf** po:buyer[  
 shipTo **hasValue** BuyerAddress **memberOf** loc:address,  
 billTo **hasValue** BuyerAddress **memberOf** loc:address,  
 hasPayment **hasValue** Payment **memberOf** po:creditCard,  
 purchaseIntention **hasValue** Trip

] and

Trip **memberOf** tc:trainTrip[  
 start **hasValue** Start,  
 end **hasValue** End,  
 departure **hasValue** Departure

]and

(Start.locatedIn = austria or Start.locatedIn = germany) and  
 (End.locatedIn = austria or End.locatedIn = germany) and  
 dt:after(Departure,currentDate) and  
 (currentDate.date.year < Payment.expYear or  
 (currentDate.date.monthOfYear =< Payment.expMonth and currentDate.date.year = Payment.expYear))."

**assumption**

**non-functional-properties**

**dc:description** "the account of the credit card has to hold a sufficient amount of money to pay the ticket."

**logical-expression**

**comment:** not specified as the ontological terminology is missing

**postCondition**

**non-functional-properties**

**dc:description** "the output of the service is a train trip wherefore

```

the start- and endlocation have to be in Austria or in Germany and
the departure date has to be later than the current Date."
logical-expression
"outputTrip memberOf tc:trainTrip[
  start hasValue Start,
  end hasValue End,
  departure hasValue Departure
] and
(Start.locatedIn = austria or Start.locatedIn = germany) and
(End.locatedIn = austria or End.locatedIn = germany) and
dt:after(Departure,currentDate)."
```

```

effect
non-functional-properties
  dc:description "there shall be a trade for the train trip of the postcondition"
logical-expression
"someTrade memberOf po:trade[
  items hasValues outputTrip,
  payment hasValue AcceptedPayment memberOf po:creditCard
]."
```

```

interface
non-functional-properties
  dc:description "describes the Interface of Web Service"
  comment: not specified yet.
```

As in the modeling in the Goal, here an instance of the concept 'Itinerary' is used as the value of the property 'Items' of the concept 'Trade', while Itinerary is not defined in tc.wsml as a subconcept of po:product. The OO-mediator that imports the terminology required for the capability and performs this operation will be included in the next version of this deliverable.

## Mediators

### OO-Mediators

OO Mediators "connect" ontologies with other ontologies or OO Mediators for refining ontologies, as well as for importing ontologies as the terminology definitions into other WSMO components. As the Goal and the Web Service specified above have homogeneous information spaces, we only have to specify OO Mediators for the existing ontologies used for the domain ontologies defined in this use case. Here, we have to define the following OO Mediators, specified in the Listings below:

1. owlAddressMediator.wsml
2. owlCurrencyMediator.wsml
3. owlFactbookMediator.wsml
4. owlPersonMediator.wsml

Listing 7: OO-Mediator " importing the OWL Address Ontology to the Location Ontology"

```

ooMediator
http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/owlAddressMediator.wsml

namespace
dc=http://purl.org/dc/elements/1.1#,
wsmo=http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418/#
```

**non-functional-properties**

**dc:title** "OO Mediator importing the OWL Factbook ontology to WSMML"  
**dc:creator** "DERI International"  
**dc:publisher** "DERI International"  
**dc:contributor** "Axel Polleres"  
**dc:date** "2004-06-07"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/#L3962>  
**dc:format** "text/plain"  
**dc:language** "en-us"  
**dc:rights** <http://deri.at/privacy.html>  
**version** "\$Revision: 1.3 \$"

**sourceComponent**

**ontology** <http://daml.umbc.edu/ontologies/ittalks/address/>

**targetComponent**

**ontology**  
<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/loc.wsml>

**mediationService**

**comment:** not yet implemented.  
**comment:** should have a full wsml-webservice description, here we only  
**comment:** give the intended endpoint of the future service  
**comment:** <http://138.232.65.151:8080/TranslatorService/OWL2WSML/>  
  
**comment:** This source ontology might overlap with the owl person ontology.  
**comment:** Not yet checked. In case, we should have one mediator importing both  
**comment:** and resolving possible overlaps/conflicts.

Listing 8: OO-Mediator "importing the OWL Currency Ontology into the Purchase Ontology"

**ooMediator** <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/owlCurrencyMediator.wsml>

**namespace**

**dc:**<http://purl.org/dc/elements/1.1/#>,  
**wsml:**<http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418/#>

**non-functional-properties**

**dc:title** "OO Mediator importing the OWL Currency ontology to WSMML"  
**dc:creator** "DERI International"  
**dc:publisher** "DERI International"  
**dc:contributor** "Holger Lausen"  
**dc:date** "2004-06-07"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/#L3962>  
**dc:format** "text/plain"  
**dc:language** "en-us"  
**dc:rights** <http://deri.at/privacy.html>  
**version** "\$Revision: 1.3 \$"

**sourceComponent**

**ontology** <http://www.daml.ecs.soton.ac.uk/ont/currency.daml>

**targetComponent**

**ontology** <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/po.wsml>

**mediationService**

**comment:** not yet implemented.  
**comment:** should have a full wsml-webservice description, here we only  
**comment:** give the intended endpoint of the future service  
**comment:** <http://138.232.65.151:8080/TranslatorService/OWL2WSML/>

Listing 9: OO-Mediator "importing the OWL Factbook into the Location Ontology"

**ooMediator**

<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/owlFactbookMediator.wsml>

**namespace**

dc=<http://purl.org/dc/elements/1.1/>#,  
wsm1=<http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418/#>

**non-functional-properties**

**dc:title** "OO Mediator importing the OWL Factbook ontology to WSML"  
**dc:creator** "DERI International"  
**dc:publisher** "DERI International"  
**dc:contributor** "Axel Polleres"  
**dc:date** "2004-06-07"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/#L3962>  
**dc:format** "text/plain"  
**dc:language** "en-us"  
**dc:rights** <http://deri.at/privacy.html>  
**version** "\$Revision: 1.4 \$"

**sourceComponent**

**ontology** <http://www.daml.org/2003/09/factbook/factbook-ont/>

**targetComponent**

**ontology**  
<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/loc.wsml>

**mediationService**

**comment:** not yet implemented.  
**comment:** should have a full wsm1-webservice description, here we only  
**comment:** give the intended endpoint of the future service  
**comment:** <http://138.232.65.151:8080/TranslatorService/OWL2WSML/>

### Listing 10: OO-Mediator "importing the OWL Person Ontology into the Train Connection Ontology"

**ooMediator** <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/owlPersonMediator.wsml>

**namespace**

dc=<http://purl.org/dc/elements/1.1/>#,  
wsm1=<http://www.wsmo.org/2004/d16/d16.1/v0.2/20040418/#>

**non-functional-properties**

**dc:title** "OO Mediator importing the OWL Person ontology to WSML"  
**dc:creator** "DERI International"  
**dc:publisher** "DERI International"  
**dc:contributor** "Axel Polleres"  
**dc:date** "2004-06-07"  
**dc:type** <http://www.wsmo.org/2004/d2/v0.3/20040329/#L3962>  
**dc:format** "text/plain"  
**dc:language** "en-us"  
**dc:rights** <http://deri.at/privacy.html>  
**version** "\$Revision: 1.3 \$"

**sourceComponent**

**ontology** <http://daml.umbc.edu/ontologies/ittalks/person/>

**targetComponent**

**ontology** <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/tc.wsml>

**mediationService**

**comment:** not yet implemented.  
**comment:** should have a full wsm1-webservice description, here we only  
**comment:** give the intended endpoint of the future service  
**comment:** <http://138.232.65.151:8080/TranslatorService/OWL2WSML/>

**comment:** This source ontology might overlap with the owl address ontology.  
**comment:** Not yet checked. In case, we should have one mediator importing both  
**comment:** and resolving possible overlaps/conflicts.

Notice that the mediation services are not specified. For importing an OWL ontology into a WSML ontology, it is obvious that such mediation services are required. The terminology to express the capability of mediation services as well as the requester goals is not defined at the moment. This terminology, modeling the ontology mediation domain, has to be included in future versions of the deliverable and the necessary goals and capabilities have to be defined using such terminology.

### ***WG-Mediators***

A WG Mediator links a Web Service to a Goal, resolves terminological mismatches, and states the functional difference (if any) between both. The main application of WG Mediators is handling of partial matches within Web Service discovery. For resolving terminological mismatches, OO Mediators are applied, similar to the ones specified above. The functional difference is stated in the reduction which restricts the set of valid ontology objects to be passed between the Web Service and the Goal.

In our use case, we do not need an WG Mediator, because the Goal and the Web Service Description use the same domain ontologies (i.e. there are not terminology mismatches), and there is no functional differences between the Goal and the Capability. An WG Mediator with a reduction would be needed if the Web Service Capability specifies that train tickets as well as plane tickets are sold: Therefore, the reduction would restrict the valid set of information to train tickets, as requested by the Goal.

### ***GG-Mediators***

A GG Mediator connects Goals, specifying the possible functionality reduction. For example, a GG Mediator would connect a Goal "buy a ticket" with another Goal "buy a train ticket" by stating the ontological correspondance between the Goals as a reduction. If 'train ticket' is a subclass of 'ticket', than the reduction in the GG Mediator would specify that valid instances for the second Goal have to be 'train ticket **subclassof** ticket'.

There is no GG Mediator needed in our use case.

### ***WW-Mediators***

A WW Mediator connects Web Services used by another Web Service in ther Orchestration, resolving heterogeneities at all levels (data, process, protocol). Also, WW Mediators are applied to resolve heterogeneities on the data, process, and protocol level between the Choreographies of Web Services that are ought to interact in a global interaction model.

There is no WW Mediator in this use case at this point in time, since the Orchechstration of the ÖBB end-user Web Service is not specified in this version. Future versions might include the definition of the composition of the search and buying service, wherein a WW Mediator might be applied.

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

On basis of the models for the WSMO components specified above, we can define automated mechanisms for 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, restricted to Web Service Discovery at this point in time.

## **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 inspects 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 as top level building blocks. 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 the realization of Web Service Discovery within FLORA-2 on basis of the use case models as specified above.

In general, Web Service Discovery is 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 basically has to be proven that the Capability logically entails the Goal with the premise that the conditions for successful usage of the Web Service are fulfilled at invocation time.

Goal-Capability-Matching is considered as the heart of Web Service Discovery, as it determines whether some Web Service can satisfy the Goal at all. Upon this, different "Discoverers" can be build that modify the discovery results with regard to specific requirements for the application. We discuss this in more detail below.

### **2. Handling partial matches in 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 states where the postconditions and effects of the Web Service are fulfilled the Goal is satisfied. This might not hold for many cases, as there might be semantic differences between the Goal and the Capability; but a Web Service might be usable for solving Goal when the valid set of outputs (as well as the set of halting states) 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**

Goal-Capability matching 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.

The theory of Web Service Discovery is exhaustively discussed in [\[Keller et al., 2004\]](#). Therein, the general proof obligation is defined, and different approaches by means of logic programming are presented as discussed in detail. For Web Service Discovery in this use case, we implemented the approach "Goal as ground facts", see section 4.2.1 in [\[Keller et al., 2004\]](#). Summarizing, the matchmaking works as follows:

- The Goal is defined as a ground fact, meaning it is a logical expression without variables. The Goal is only specified partially, meaning that concrete values are only specified for the subset of the properties of the ontological concepts that is needed to express the desire; e.g., for the itinerary that a ticket is desired for, no information is specified for 'arrival' (see Listing 5) .
- The description elements of the Web Service Capability are modeled as rules. Here also, only the subset of properties of the ontological notions that is needed to describe the information structure and conditions is specified; besides, the body of each rule contains conditions that restrict the range of valid values.

The approach for matchmaking is to check whether the ground facts specified in the Goal satisfy the rules of the Capability. Obviously, this is facilitated by the construction of Goals and Capabilities. Considering the Goal as specified in Listing 5, the Goal postcondition is fact that satisfies the body of the Capability postcondition in Listing 6: the Goal Postcondition is a specific itinerary (from Innsbruck to Frankfurt with a certain departure), and the body of the Capability postcondition requires an itinerary with start- and endlocation in Austria or Germany, respectively, and with a departure that has to be later than the current date (the current date is modeled as an instance of `dateandtime` from the Date and Time Ontology as there is no built-in function yet). The same matching holds for the Goal Effect in correlation to the Capability Effect. Thus, the following discovery query returns the identifier of the ÖBB end-user service, as a service that can satisfy the Goal. The result of successful discovery in Flora is shown in the screenshot in Figure 7.

```
?- Webservice:webservice[
    capability->_C:capability[postcondition,effect]
].
```

```

/cygdrive/c/_projects/_WSMO/_cvs/wsmo/d3/d32/resources
[FLORA: Compiling c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.flr]
[Preprocessing c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.flr]
[FLORA: Compile time 0.2800 seconds]
[FLORA: Dependency checking time: 0.1100 seconds]
[FLORA: [Coder] Generating P file 0.0200 seconds]
[FLORA: [Coder] Generating DB file 0.0100 seconds]
[FLORA: Done! CPU time used: 0.4500 seconds]
[FLORA: Loading c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.flr into mod
ule main]
[Compiling c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match]
[Preprocessing c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.P]
[match compiled, cpu time used: 2.5640 seconds]
[match_main loaded]
[FLORA: Dynamically loading c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.
fdb into module main]
[Preprocessing c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.fdb]
[FLORA: Done! CPU time used: 0.0200 seconds]
[FLORA: Dynamically loading c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.
fld into module main]
[Preprocessing c:\_projects\_WSMO\_cvs\_wsmo\d3\d32\resources\match.fld]
[FLORA: Done! CPU time used: 0.0000 seconds]

Testing if the KB is inconsistent:
No

Querying for Web Services for the goal in goal.flr:
Webservice = oebbWS

1 solution(s) in 0.0100 seconds on NB-HL

Yes

Yes

flora2 ?-

```

Figure 7. Successful Web Service Discovery in FLORA-2

The transformation of the WSMO models as specified in the Listings to FLORA-2 compilable F-Logic is trivial, as WSML relies on F-Logic. The complete models for the use case as FLORA-compilable resources is provided in [Appendix A](#). However, there are some aspects that have to be considered for the transformation of WSML definitions to FLORA-2 syntax:

1. **Signature Checking** Class Signatures (single and multi-valued attributes only) are checked via a set of rules, that can be found in the source file flr.flr in the appendix. Note that there is yet no agreed syntax for f-logic to define the signatures of relation or method parameters, therefore we currently do not check those signatures. We refer to the ongoing discussion in the f-logic syntax consortium recently established by Ontoprise, Michael Kifer and DERI.
2. **Integrity Constraints** In WSML they are modelled as rules with empty head, flora does not have a build in support to check them, therefore we use the two valued relation invalid in the head of the rule, the first parameter is the instance that is violating some constraint, the second is a String that identifies the constraint which is violated. The integrity of the knowledge base is checked the query "?-invalid(X,Y)", only if this query returns no values the knowledge base has a model.
3. **Mediators** are not implemented yet, all flora files are simply imported in one single file and then evaluated.
4. **Namespaces** are not implemented.

5. **Non Functional Properties** are not translated directly into the ontologies represented in flora2 syntax. They are only present as inline comments.
6. **Evaluation of Arithmetic Expressions** is not always done in flora2 unless the "is" operator is used, therefore some "=" in the WSML listings have been changes to "is".
7. **Function Symbols** have not been defined directly as function symbols in the flora files, but rewritten as relations with an arity of n+1 (w.r.t. arity of the function), see secondsAfterMidnight relation in the dt.flr for an example.
8. **Precision of Floating Point Operations** Due to the imprecise handling of floating point arithmetics in the XSB engine underlying flora2, some axiomatizations given in the wsml file have been implemented differently. See for example the julianDayNumber function.
9. **Variable Declarations** have been omitted int the flora2 files, a variable is simply identified as a upper case term.
10. **Redefinition of Build-In Predicates** is not allowed in flora2, therefore new relations are defined, for example equal(X,Y) which is equivalent to X=Y in the wsml files.

In conclusion, the approach for Goal-Capability-Matching presented here seems to be a promising solution because it is heading towards the right direction. For further development of the Goal-Capability-Matching technology within WSMO as the heart of Web Service Discovery, very complex and challenging efforts have to faced, e.g. definition of a decidable subset for the specifications of logical expression in WSML as well as implementation of logical entailment for the reasoner to be supported within WSMO.

### 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 top-level 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 a Goal in the Use Case, with respect to the description elements defined for Goals in WSMO
- **Web Service Capability:** We have derived a specification on how to model Web Service Capabilities
- **Mediators:** We have modeled the needed WSMO Mediators needed for the Use Case, although the mediator service specification is still open.

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 WSMO, in particular

- 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 the specification and realization of Goal-Capability Matching as the heart of WSMO-enabled Web Service Discovery mechanisms. 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 showcasing further WSMO constructs.

## 4. Conclusions and Further Work

Apart 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 WSML 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, with special attention to Goal-Capability-Matching as the heart of Discovery.

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 Integratin 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

**[Arroyo et al., 2004]** Arroyo, S.; Lara, R.; Gómez, J.; Berka, D.; Ding, Y.; Fensel, D. (2004): Semantic Aspects of Web Services. In Munindar. P. Singh (Ed.), Practical Handbook of Internet Computing. Baton Rouge: Chapman Hall and CRC Press, Baton Rouge. 2004.

**[Arroyo and Stollberg, 2004]** Arroyo, S.; Stollberg, M.: *WSMO Primer*. WSMO Deliverable D3.1, available at: <http://www.wsmo.org/2004/d3/d3.1/v0.1/>

**[Biron & Malhotra, 2001]** P. V. Biron and A. Malhotra: *XML Schema Part 2: Datatypes*, W3C Recommendation 02, 2001, available at: <http://www.w3.org/TR/xmlschema-2/>.

**[Bray et al. (Eds.), 1999]** Bray, T.; Hollander, D. and Layman, A.: *Namespaces in XML*, W3C Recommendation, 14 January 1999, available at: <http://www.w3.org/TR/REC-xml-names/>.

**[EDIFACT]** UN/EDIFACT, available at: <http://www.unece.org/trade/untdid/welcome.htm>.

**[Fensel & Bussler, 2002]** D. Fensel and C. Bussler: *The Web Service Modeling Framework WSMF*, Electronic Commerce Research and Applications, 1(2), 2002.

**[He et al., 2004]** he, H.; Haas, H.; Orchard, D.: *Web Services Architecture Usage Scenarios*, W3C Working Group Note 11 February 2004. available at: <http://www.w3.org/TR/ws-arch-scenarios/>.

**[Hayes (Ed.), 2004]** P. Hayes: *RDF Semantics*, W3C Recommendation 10 February 2004, 2004.

**[Keller et al., 2004]** Keller, U.; Lara, R.; Polleres, A.; Lausen, H.; Stollberg, M., Kifer, M.: *Inferencing Support for Semantic Web Services: Proof Obligations*. WSML Deliverable D5.1 v0.1, WSML Working Draft 21 June 2004. available at <http://www.wsmo.org/2004/d5/d5.1/v0.1/20040621/>

**[Kifer et al., 1995]** M. Kifer, G. Lausen, and James Wu: *Logical foundations of object oriented and frame-based languages*. Journal of the ACM, 42(4):741-843, 1995.

**[Pan and Hobbs]** F. Pan and R. Hobbs: *Time in OWL-S*, available at: <http://www.isi.edu/~pan/damlttime/AAAsymp2004.pdf>.

**[Oren et al, 2004]** Oren, E. (Ed.): *BNF grammar for WSML language.*, WSMO Deliverable D16.1 v0.2, Working Draft 12 June 2004 available at: <http://www.wsmo.org/2004/d16/d16.1/v0.2/20040612/>.

**[Roman et al., 2004]** D. Roman, U. Keller, H. Lausen (eds.): *Web Service Modeling Ontology - Standard (WSMO - Standard)*, version 0.2 available at <http://www.wsmo.org/2004/d2/v02/>

**[SOAP]** Mitra, N.: *SOAP Version 1.2 Part 0: Primer*. W3C Recommendation 24 June 2003. available at: <http://www.w3.org/TR/soap12-part0/>

**[Thompson et al. (Eds.), 2001]** H. S. Thompson, D. Beech, M. Maloney and N.Mendelsohn: *XML Schema Part 1: Structures*, W3C Recommendation, May 2001, available at: <http://www.w3.org/TR/xmlschema-1/>.

**[UDDI]** Bellwood, T.; Clément, L.; von Riegen, C. (Ed.): *UDDI Version 3.0.1*. UDDI Spec Technical Committee Specification, Dated 20031014. available at: [http://uddi.org/pubs/uddi\\_v3.htm](http://uddi.org/pubs/uddi_v3.htm)

**[WSDL]** Chinnici, R.; Gudgin, M.; Moreaum, J.-J.; Weerawarana, S. (2003): *Web Services Description Language (WSDL) Version 1.2*. W3C Working Draft 3 March 2003. available at <http://www.w3.org/TR/wsdl20/>.

**[Yang and Kifer, 2003]** G. Yang, M. Kifer: *Reasoning about Anonymous Resources and Meta Statements on the Semantic Web* J. Data Semantics I 2003: 69-97.

## Acknowledgements

The work is funded by the European Commission under the projects [DIP](#), [Knowledge Web](#), [SEKT](#), [SWWS](#), [Esperanto](#), 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 WSMO working group](#) for their advice and input into 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 [3.1](#) can be download as computational resources for testing and development of WSMO with FLORA-2, an F-Logic reasoner. The files below contain the WSMO 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 under ongoing development . The most recent resources can be accessed via CVS web-interface at: [http://cvs.deri.at/cgi-bin/viewcvs.cgi/\\*checkout\\*/wsmo/d3/d32/resources/](http://cvs.deri.at/cgi-bin/viewcvs.cgi/*checkout*/wsmo/d3/d32/resources/).*

### Ontology 1: "International Train Connections Domain Ontology"

```
// International train connections domain ontology

//Concept definitions
station::location[
//Code of the station
  code => string,
  locatedIn =>> location,
//For stations located at the border
  borderToCountry => border
].

//Notice that this subclassing would be done by a mediator
itinerary::product[
  passenger => person,
  recordLocatorNumber => string,
  trip => trip
].

//We do not have the mediators functionality to import the OWL person ontology yet,
so we define it here based on the OWL ontology
person[
  firstName => string,
  lastName => string,
  title => string,
  homeAddress => address,
  officeAddress => address,
  email => string,
  homePhone => string,
  officePhone => string,
  cellPhone => string,
  fax => string,
  pager => string,
  homepage => string,
  gender => string,
  birthday => date
].

trip[
  start => location,
  end => location,
  via =>> location,
  departure => dateAndTime,
  arrival => dateAndTime,
  duration => interval,
  distance => distance
].

trainTrip::trip[
  start => station,
  end => station,
  via =>> station,
```

```

    seat => string,
    train => string,
    class => string
].

//Integrity constraint "stationCountry"
invalid(S,"stationCountry")
:- S:station[
    locatedIn -> L,
    country -> C],
    L:location[
    country -> C2],
    C \= C2.

//Integrity constraint "departureBeforeArrival"
invalid(T,"departureBeforeArrival")
:- T:trip[
    departure -> D,
    arrival -> A],
    (after(D,A) ; equal(D,A)).

//Integrity constraint "startNotEqualEnd"
invalid(T,"startNotEqualEnd")
:- T:trip[
    start -> Start,
    end -> End],
    Start=End.

innsbruckHbf:station[
    name->"Innsbruck Hbf",
    code->"INN",
    locatedIn ->> {innsbruck}
].

frankfurtHbf:station[
    name->"Frankfurt Hbf",
    code->"FKF",
    locatedIn ->> {frankfurt}
].

//locatedIn transetive with country...
Station[locatedIn ->> Country]
:- Station:station,
    Station..locatedIn = City:city,
    City..country = Country:location.

```

## Ontology 2: "General Date and Time Ontology"

```

#include "flr.flr"

//concept defintions 1:1 from wsml
instant[
].

interval[
    start=>instant,
    end=>instant
].

date[
    dayOfMonth=>dayOfMonth,
    monthOfYear=>monthOfYear,
    year=>year
].

dayOfMonth::integer.

```

```

year::integer.
monthOfYear::integer.

time[
  hourOfDay=>hourOfDay,
  minuteOfHour=>minuteOfHour,
  secondOfMinute=>secondOfMinute
].

secondOfMinute::integer.
minuteOfHour::integer.
hourOfDay::integer.

dateAndTime::instant[
  date=>date,
  time=>time
].

//variable defintions are skipt in the flr file!

//functionDefinitions we can define methoddefintions but no functions
//julianDayNumber
//daysBetween
//secondsBetween
//secondsFromMidnight

//relationdefintions
//not implemented for relations
//contains(parameter1:interval, parameter2:intervalOrInstant).

//axiomDefintions
invalid(X,"invalidMonthOfYear")
  :- X:monthOfYear, (X < 1; X > 12).

invalid(X, "invalidDayOfMonth")
  :- X:dayOfMonth, tnot (X < 1 , X > 31).

invalid(X, "invalidDate")
  :-
  X:date, (
    (X.dayOfMonth > 28 , X.monthOfYear = 2,
     \+((Y4 is mod(X.year ,4)@prolog(), Y4=0,
            Y100 is mod(X.year ,100)@prolog(), Y100\=0);
            Y400 is mod(X.year ,400)@prolog(), Y400=0))
    ; (X.dayOfMonth > 29 , X.monthOfYear = 2)
    ; (X.dayOfMonth > 30 , X.monthOfYear = 4)
    ; (X.dayOfMonth > 30 , X.monthOfYear = 6)
    ; (X.dayOfMonth > 30 , X.monthOfYear = 9)
    ; (X.dayOfMonth > 30 , X.monthOfYear = 11)
  ).

invalid(X,"invalidHourOfDay")
  :- X:hourOfDay, (X < 0; X >= 24).

invalid(X,"invalidMinuteOfHour")
  :- X:minuteOfHour, (X < 0; X >= 60).

invalid(X,"invalidSecondOfMinute")
  :- X:secondOfMinute, (X < 0; X >= 60).

invalid(X,"invalidInterval")
  :- X:interval, after(X.start,X.end) ; equal(X.start,X.end).

//overvridding of build in predicates not possible, therefore new predicate equal,
before, after
equal(X,Y) :-
  Y:date, X:date,
  X.dayOfMonth = Y.dayOfMonth,

```

```

X.monthOfYear = Y.monthOfYear,
X.year = Y.year.

before(X, 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)
  ).

after(X, Y) :-
  before(Y, X).

// however due to a bug in XSB (float number arithmetics do not work proper)
// this could not be implemented and a simplified version that aproximates the days
// after Christ is used.

daysAfterChrist(D,X) :-
  D:date, daysAfterBeginOfYear(Z, D.monthOfYear),
  X is (D.dayOfMonth + Z + (D.year*365)).

//since the floating point unit is not exact enough got the
//axiom we use a simplified axiomatiaztion without leap year
/*
julianDayNumber(X,JDN) :-
  X:date,
  (X.monthOfYear<3,
   Y is X.year-1,
   M is X.monthOfYear+12);
  (X.monthOfYear>2,
   Y is X.year,
   M is X.monthOfYear),
  D is X.dayOfMonth,
  A is '//'(Y,100)@prolog(),
  B is '//'(A,4)@prolog(),
  C is 2-A+B,
  E is floor(365.25*(Y+4716))@prolog(),
  F is floor(30.6001*(M+1))@prolog(),
  JDN is C+D+E+F-1524.
*/
//relation daysAfterBeginOfYear holds the number of days (argument 1) that have
passed
//at the end of the month (argument 2) from the begin of the year
//this is used to calculate differences in dates
daysAfterBeginOfYear(31, 1).
daysAfterBeginOfYear(59, 2).
daysAfterBeginOfYear(90, 3).
daysAfterBeginOfYear(120, 4).
daysAfterBeginOfYear(151, 5).
daysAfterBeginOfYear(181, 6).
daysAfterBeginOfYear(212, 7).
daysAfterBeginOfYear(243, 8).
daysAfterBeginOfYear(273, 9).
daysAfterBeginOfYear(304, 10).
daysAfterBeginOfYear(334, 11).
daysAfterBeginOfYear(365, 12).

//the difference in days between 2 dates
daysBetween(D1,D2,X) :-
  D1:date, D2:date,
  daysAfterChrist(D1,DAC_D1),
  daysAfterChrist(D2,DAC_D2),
  X is DAC_D1 - DAC_D2.

// computes if two given times are the same

```

```

equal(X,Y) :-
  X:time, Y:time,
  X.secondOfMinute = Y.secondOfMinute,
  X.minuteOfHour = Y.minuteOfHour,
  X.hourOfDay = Y.hourOfDay.

// computes if a given time X is before another time Y
before(X,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 if a given time X is after another time Y
after(X,Y)
  :- before(Y,X).

//computes the amount of seconds from midnight
secondsFromMidnight(T,X) :-
  T:time,
  X is T.secondOfMinute + (T.minuteOfHour*60) + (T.hourOfDay*60*60).

//the difference in seconds between 2 times
secondsBetween(T1, T2, X) :-
  T1:time, T2:time,
  secondsFromMidnight(T1,SFM_T1),
  secondsFromMidnight(T2,SFM_T2),
  X is SFM_T1 - SFM_T2.

//computes if Date and Time are equal
equal(X,Y) :-
  X:dateAndTime, Y:dateAndTime,
  equal(X.date,Y.date),
  equal(X.time,Y.time).

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

//computes if a given date and time X is after another date and time Y
after(X,Y) :-
  X:dateAndTime, Y:dateAndTime,
  before(Y,X).

//computes the difference in seconds between two different DateAndTime
secondsBetween(D1, D2, X) :-
  D1:dateAndTime, D2:dateAndTime,
  daysAfterChrist(D1.date,DAC_D1),
  daysAfterChrist(D2.date,DAC_D2),
  secondsFromMidnight(D1.time,SFM_T1),
  secondsFromMidnight(D2.time,SFM_T2),
  X is SFM_T1 + DAC_D1 * 24 * 60 * 60 -
  (SFM_T2 + DAC_D2 * 24 * 60 * 60).

//the difference in days between two different DateAndTime
daysBetween(D1, D2, X) :-
  D1:dateAndTime, D2:dateAndTime,
  daysAfterChrist(D1.date,DAC_D1),
  daysAfterChrist(D2.date,DAC_D2),
  X is DAC_D1 - DAC_D2.

//computes if a interval X contains a second interval Y
contains(X,Y) :-
  X:interval, Y:interval,
  (before(X.start, Y.start);equal(X.start, Y.start)),
  (after(X.end, Y.end);equal(X.end, Y.end)).

```

```

//computes if a interval X contains a instant Y
contains(X,Y) :-
  X:interval, Y:instant,
  (before(X.start, Y);equal(X.start, Y)),
  (after(X.end, Y);equal(X.end, Y)).

//The current Date manually since we do not have build in function yet
currentDate:dateAndTime[
  date->date:date[dayOfMonth->28,monthOfYear->2,year->2001],
  time->time:time[hourOfDay->23,minuteOfHour->40,secondOfMinute->12]
].

//test instance:
exampleDT0:dateAndTime[
  date->date1:date[dayOfMonth->1,monthOfYear->1,year->2001],
  time->time1:time[hourOfDay->23,minuteOfHour->40,secondOfMinute->12]
].

exampleDT1:dateAndTime[
  date->date2:date[dayOfMonth->1,monthOfYear->1,year->2001],
  time->time2:time[hourOfDay->23,minuteOfHour->40,secondOfMinute->30]
].

exampleDT2:dateAndTime[
  date->date3:date[dayOfMonth->2,monthOfYear->1,year->2001],
  time->time3:time[hourOfDay->22,minuteOfHour->40,secondOfMinute->00]
].

exampleDT3:dateAndTime[
  date->date4:date[dayOfMonth->2,monthOfYear->1,year->2001],
  time->time4:time[hourOfDay->23,minuteOfHour->40,secondOfMinute->12]
].

int1:interval[
  start->exampleDT0,
  end->exampleDT3
].

int2:interval[
  start->exampleDT1,
  end->exampleDT2
].

```

### Ontology 3: "Purchase Ontology"

```

buyer[
  shipTo => address,
  billTo => address,
  purchaseIntention =>> tradeItem,
  hasPayment =>> paymentMethod
].

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

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

product[
  name => string

```

```

].

price[
  amount => float,
  currency => currency
].

//The concept currency would be taken from another ontology. As it is a DAML
ontology and mediators are not implemented, we put it (partially) here
currency[
  country => country,
  name => string
].

paymentMethod[
  name => string
].

trade[
  items =>> tradeItem,
  buyer => buyer,
  seller => seller,
  payment => paymentMethod
].

delivery[
  products =>> product,
  receiver => buyer,
  sender => seller
].

creditCard::paymentMethod[
  holder => string,
  expMonth => monthOfYear,
  expYear => year,
  type => string
].

cash::paymentMethod[
  currency => currency
].

```

#### Ontology 4: "Locations Ontology"

```

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

//Mediators not implemented yet, so we do not define it as subconcept of the OWL
ontology for countries
country::location[
  name => string,
  fipsCode => string,
  isoCode => string
].

//Mediators not implemented yet, so we do not define it as subconcept of the OWL
ontology for addresses
address[
  roomNumber => string,
  streetAddress => string,
  city => city,
  state => state,
  zip => string,
  country => country

```

```

].

city::location[
  state => state,
  population => integer,
  extension => integer,
  zipcodes =>> string
].

state::location[
  cities =>> city,
  population => integer,
  extension => integer
].

border::location[
  countryA => country,
  countryB => country
].

distance[
  amount => float,
  units => string,
  kilometers => float,
  miles => float
].

//Integrity constraint "invalidDistance"
invalid(D,"invalidDistance")
:- D:distance[
  amount -> A,
  units -> U],
  tnot (A >= 0; U="Kilometers"; U="Miles").

//Calculation of a distance in kilometers
D[kilometers -> K]
:- D:distance, D[amount->A, units->U],
  ( (U="Kilometers", K is A) ;
    (U="Miles", K is A*1.609344)).

//Calculation of a distance in miles
D[miles -> M]
:- D:distance, D[amount->A, units->U],
  ( (U="Kilometers", M is A / 1.609344) ;
    (U="Miles", M is A)).

//Equality of distances
equal(D1, D2) :- D1:distance, D2:distance, D1.kilometers = D2.kilometers.

//Less than for distances
lessThan(D1, D2) :- D1:distance, D2:distance, D1.kilometers < D2.kilometers.

//More than for distances
moreThan(D1, D2) :- D1:distance, D2:distance, D1.kilometers > D2.kilometers.

germany:country[name -> 'Germany'].
austria:country[name -> 'Austria'].

innsbruck:city[name ->'Innsbruck', country ->> austria].
frankfurt:city[name ->'Frankfurt', country ->> germany].
kufstein:city[name -> 'Kufstein', country ->> austria].
salzburg:city[name -> 'Salzburg', country ->> austria].

```

**Goal:** "buying a train ticket from Innsbruck to Frankfurt on May 17th, 2004, starting at 6 p.m."

```

goal[
  postcondition=>fact,
  effect=>fact
].

mygoal:goal.

mygoal[postcondition->
  someItinerary:itinerary[
    trip -> someTrip:trainTrip[
      start -> innsbruckHbf,
      end -> frankfurtHbf,
      departure -> _#:dateAndTime[
        date -> _#:date[
          dayOfMonth->17,
          monthOfYear->7,
          year->2004
        ],
        time-> #:time[
          hourOfDay->18
        ]
      ]
    ],
    passenger->_#:person[firstName -> "Tim", lastName -> "Berners-Lee", email ->
"timbl@w3.org"]
  ]
].

// Goal Effect
mygoal[effect->
  sometrade:trade[
    items ->> someTrip,
    buyer -> _#:buyer[
      shipTo -> timsAddress:address[
        roomNumber -> 3,
        streetAddress -> 'Tims street',
        city -> boston,
        state -> massachusetts,
        zip -> 02103
      ],
      billTo -> timsAddress
    ],
    payment-> _#:creditCard[
      holder -> 'Tim Berners-Lee',
      expMonth -> 9,
      expYear -> 2007,
      type -> 'MasterCard'
    ]
  ]
].

```

**Web Service:** "selling international train tickets online for Austria and Germany"

```

// defining a webservice:
webservice[
  capability=>capability
].

oebbWS:webservice[
  capability->oebbCap:capability
].

// define an instance of Web Service Capability

```

```

oebbCap[precondition]
:-
  _InputBuyer:buyer[
    shipTo->BuyerAddress:address,
    billTo->BuyerAddress:address,
    hasPayment->Payment:creditCard,
    purchaseIntention->Trip
  ],
  Trip:trainTrip[
    start -> Start,
    end -> End,
    departure -> Departure
  ],
  (Start..locatedIn=austria ; Start..locatedIn=germany),
  (End..locatedIn=austria ; End..locatedIn=germany),
  after(Departure,currentDate),
  (currentDate.date.year < Payment.expYear;
   (currentDate.date.monthOfYear =< Payment.expMonth,
    currentDate.date.year=Payment.expYear)).

// Postcondition: the output of the service is a ticket (train ticket) for an
itinerary wherefore
// the start- and endlocation have to be in Austria or in Germany and
// the departure date has to be later than the current Date
// and binding the Trip to a predicate, so we can reference it from the effect
oebbCap[postcondition], trip -> Trip :-
  Trip:trainTrip[
    start -> Start,
    end -> End,
    departure -> Departure
  ],
  (Start..locatedIn=austria ; Start..locatedIn=germany),
  (End..locatedIn=austria ; End..locatedIn=germany),
  after(Departure,currentDate).

// Effect: there shall be a trade for the train ticket of the postcondition
oebbCap[effect] :-
  trip -> Trip,
  _SomeTrade:trade[
    items ->> Trip,
    payment -> Payment
  ],
  Payment:creditCard.

```

We do not provide the Mediators here, as they would have to be translated to Flora-specific technology.

For testing the Goal-Capability-Matching for this use case within FLORA-2, you have to download all the files above into a directory. Then, you have to set up the right environment for running the example. Therefore we provide the following additional resources, which facilitate the representation of WSMO models in Flora2, as described in Section [3.1.3](#). please note that Flora is sometimes "buggy", and you should make sure that all the resources are compiled with their latest status before you run the Goal-Capability-Matching program. It also sometimes happen that you do not receive the correct answer for some arbitrary runtime reasons:

- [make.sh](#): this is a shell script that cleans the directory and pre-compiles all resources.

```
rm *.fld
rm *.P
rm *.xwam

runflora -e [match].
```

- [flr.flr](#): FLORA-specific (checks signatures, etc.)

```
// ##### FLORA SPECIFIC #####
// 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

// This is actually questionable. Should this treated relaxed or
// strict, i.e. is a fact invalid when it does not explicit state
// that one of the attribute values belongs to the coressponding range defintion
// or should it be infered that the attribute value is member of the range given in
// the signature?

Y:Range :-
  X:Class, X[Attribute->Y], Class[Attribute=>Range].

Y:Range :-
  X:Class, X[Attribute->>Y], Class[Attribute=>>Range].

////////////////////////////////////
// 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

// First we check if there EXISTS a signature for the Attribute (in the class
// or in any superclass)
X[valid(Attribute)] :-
  X:Class, X[Attribute->_Value], Class[Attribute=>_Range].

X[valid(Attribute)] :-
  X:Class, X[Attribute->>_Value], Class[Attribute=>>_Range].

//Now we check FORALL Attributes if there exists Range Decleration
invalid(X, Y) :-
  X:Class, X[Attribute->_Value], tnot X[valid(Attribute)],
  Y = 'no signature for' +Attribute+ 'in Class' + Class.

invalid(X, Y) :-
  X:Class, X[Attribute->>_Value], tnot X[valid(Attribute)],
  Y = 'no signature for' +Attribute+ 'in Class' + Class.

//checking if an integer is really an integer
invalid(X,Y) :-
  X:integer, tnot integer(X)@prolog(),
  Y = X+' is no integer'.

//checking if a float is really a float
invalid(X,Y) :-
  X:float, tnot float(X)@prolog(),
  Y = X+' is no float'.

////////////////////////////////////
// Integrity Constraint for relations:
// not implemented!

////////////////////////////////////
//We can check invalid(X) or invalid(X,Y) depending if you want to have verbose
error or not
```

```
invalid(X) :- invalid(X,_Y).
```

- [match.flr](#): this file includes all the resources and throws the query for Goal-Capability-Matching.

```
//includes flora specific stuff like signature checking
#include "flr.flr"
//including resources
#include "dt.flr"
#include "po.flr"
#include "tc.flr"
#include "loc.flr"

#include "ws.flr"
#include "goal.flr"

// checking if the kb is valid:
// in case this query gives an answer the kb is inconsistent
// and the discovery process should be aborted
?- write('\nTesting if the KB is inconsistent:')@prolog(), invalid(X,Y).

//this query checks if given the kb imported above
//includes a suitable service for the goal given in goal.flr
?- write('Querying for Web Services for the goal in goal.flr:')@prolog(),
  Webservice:webservice[
    capability->_C:capability[postcondition,effect]
  ].
```

---

## Appendix B: Change Tracking

To facilitate retracing of changes inbetween different version of this deliverable, the following lists the essential changes done in comparison to the preceding version.

The change tracking starts with the version of 28 June 2004.

**Version: 28 June 2004** <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/>

- complete read-thru with corrections of deliverable text (regarding comments from Jos de Bruijn)
- corrections of domain ontologies
  - \* changed section 3.1.1 to "Use Case Overview", describes the properties of the WSMO components modeled below
  - \* the web service described now is understood as an aggregated / composed web service that offers the overall functionality for purchasing train tickets online. In later versions, the Choreography description as well as the Orchestration with specific Web Services for searching and buying train tickets can be adopted.
  - \* corrected / clarified descriptions for modeling descriptions.
- correction of WSML-models for Goals, Web Services, Mediators
- revised the Web Service Discovery description (section 3.1.3)
- updated the FLORA2 resources to the WSML models (as in Listings)

- namespace handling refined

---

## Appendix C: Review Comments

1. Sometimes, references are still lacking. For example, page mentions at the top WSMO Standard v0.2, the 2004-04-18 draft of D16.1 and the FLORA-2 homepage without proper references. There do appear to be links in the html version, but this is of course not enough.  
[Rubén Lara] References have to be added!!!  
[MS] done.
2. You sometimes mention WSMO v0.2 and then v0.3. Which version are you using?  
[Rubén Lara] v0.2 (we have to use a final stable version)
3. on page 15 (section 3.1) at the bottom you mentions two web services. These web services should be described in a bit more detail.  
[Rubén Lara] Michael, can you take care?  
[HL] Indeed. Also it has to be clarified if we include more details, e.g.: <http://138.232.65.151:8080/axis/services/scrapper?wsdl> But I think better no, since we wont finish the alignment until Monday...  
[MS] the section 3.1.1 is changed from "Requirements Analysis" to "Use Case Overview ", which gives a concise overview of the WSMO components defined for the use case. done.
4. section 3.1.1: Functional Requirements does \*not\* list requirements and certainly does not do a requirements analysis, as is claimed in the first line of the first paragraph of the section. There is merely a paragraph which says that a requirements analysis will follow.  
[Rubén Lara] This is in fact a description of the use case. I have updated the text and the table headings
5. page 17/18: there are unreferenced tables (tables 1-4) are here and they seem to belong to section 3.1.1, because they have the word "requirements" in the captions. Furthermore, the tables seem to list properties of the use case, instead of requirements coming from the use case.  
[Rubén Lara] Updated
6. Page 17, table 1, IMO, this table should be split up in different tables for each ontology that has been identified.  
[Rubén Lara] We keep it like this until we have a stable set of ontologies needed
7. page 17/18, tables 2,3,4, you describe a goal, web services, and mediators, instead of requirements on goal modeling, WS modeling and mediator modeling  
[Rubén Lara] Heading updated. Now it is "properties", not "requirements"
8. Page 18, first paragraph of 3.1.2: it's not really clear what you want to say here. Do you want to describe limitations of WSMO? Furthermore, again there is no reference for WSMO-Standard and here you say that you use v0.3, whereas later on in this paragraph you say you use v0.2  
[Rubén Lara] Updated

9. Section 3.1.2, pages 18/19: your description of namespaces seems more like a description of the use of qualified names. I think you should split up the bullet point and make two bullets, one for namespaces and one for qnames. Furthermore, in this paragraph you fail to say that WSMO keywords can be identified in the listings by being in boldface. You should in the namespace bullet (or maybe you need a separate subsection) also explain how namespaces are used, how the target namespace is used (the feature new in WSML since yesterday's new release of d16.1).  
[HL] Resolved, cf bullet 1 & 2 in section 3.1.2
10. Section 3.1.2, page 19, bullet on function symbols: you should make clear in this bullet how function symbols in wsml relate with function symbols and predicates in logics.  
[HL] predicates in logics are relations in wsml; function symbols in WSML are intentionally the same as in logics, I do not see which explanations are missing. Bear in mind that the section only should give an overview of the modeling, space for definitions of precise semantics is elsewhere. Furthermore additional information can be found in the section that describes the relation between flora2 and wsml
11. Section 3.1.2, page 19, bullet on literal vs. resource: you say "literals are enclosed by quotes". however, not all literals in the listings are enclosed in quotes, more specifically, I haven't seen any integers enclosed in quotes. I propose not to require quotes for integers.  
[HL] This was already mentioned in the last paragraph of this bullet (For brevity, in the listings we allow an abbreviated notation...)
12. General question on data types in wsml: are they ordered? Are they closed under negation?  
[HL] This depends on the data type. XSD mentions "facets" on data types that define special predicates per data type, such a facet could be for integer having a predicate "<" defining a total order of the set. Again the complete definition of this mechanism is not in the scope of a use case document, we just try to sketch a possible approach.
13. Another general question: what data types are you using? You mention xsd, but don't really say that you use the xml schema data types. You should be a bit more clear here.  
[HL] a list is provided now.
14. Is "-" a binary or a unary function symbol?  
[HL] Both. As mentioned in the paragraph on Literal vs. Resources
15. What does this "1.23"^^xsd:double mean? Are you adhering to this proposal for data types in RDF by Pat? Why?  
We are adhering to the encoding of that for the moment. But we do not limit ourselves to the semantics of Pat that do not consider facets of data types such as ordering. However alternative proposals can be examined (encoding and semantics). However for this version we stick to Pat's syntax and provide a sketch of an extended semantics. Full definition of that is not in the scope of a use case document.
16. On page 23 you should explain more clearly what the relationship is between XSD and your date/time ontology, because there's already date and time in xsd.

[HL] We do allow encoding at the moment only according to the date and time ontology format and not as xsd:date.

17. Page 30, third bullet: you must mean EDIFACT instead of EDI? Also, please provide a reference

[Rubén Lara] I think he is right. Updated.

18. Page 33, bottom: In the postcondition I would also expect some kind of confirmation of the purchase, because it is a change in the information space

[Rubén Lara] I would suggest to skip it at the moment.

[HL] I think that is covered by the fact that a ticket exists on the customers name.

[JB] I agree with RL that it should be skipped for now, but a remark should be made that the description is not complete

[MS] In fact, such a confirmation / contract of purchase would be an essential construct in a general purchase ontology. It was decided that such an enhanced purchase ontology would be a collaborative effort of different projects / working groups - in fact, a general purchase ontology is as essential as a general date and time ontology.

19. Page 35: you say here that you only describe one web service, whereas earlier in section 3 you were talking about two web services. What's it going to be?

[Rubén Lara] Michael, can you do it consistent? [HL] Identical to comment No 3 therefore closed.

20. Page 39/40. Are you talking here about a language for expressing mappings between ontologies or for describing the capability of a web service?

[Rubén Lara] Capability. In fact, it is written in the text of the current version...

[MS] the concept for OO Mediators (as for WSMO Mediators in general) is not yet specified in WSMO Standard. The whole Mediators section will be updated accordingly when we have a stable specification for this

21. Page 42, second paragraph, last sentence: you say "Obviously, the Proof Obligation is fulfilled". This is not so obvious to me. Please elaborate.

[Rubén Lara] This whole section will be deleted and a reference to D5.1 added.

22. Page 44, bottom: What do you mean with that logical entailment is not satisfiable in FOL?

[HL] Who solved that one? Michael?

[MS] whole section is revised. the "theoretical" part of web service discovery / goal-capability-matching is now provided in WSMO D5.1. This section now only briefly explains the realization in FLORA, with the "goals as ground facts" - approach.

23. Page 44, bottom: you could also mention here that in the case of DL, the matching problem could probably be reduced to a subsumption problem.

[HL] I propose to ignore this one and shift to 5.1?

[MS] same as for no. 22.

24. Page 45, top: it would be useful to have some hints here on the second phase of the discovery, where the user has to decide whether he can fulfill the preconditions and whether the assumption are valid.

[HL] future versions, or is it resolved?

[MS] (1) this is this for future versions, (2) the theory and approach for web service discovery is concentrated in WSMO D5.1

25. First of all some comments on the use of namespaces: You should adopt the use of the `targetNamespace`, which specifies the namespace in which the elements are defined (cf. `targetNamespace` in XML Schema)  
[Rubén Lara] I think he is right. Comments?  
[HL] I.e. that means we have two "default" namespaces as in XML, whenever you reference an existing element the default namespace is used, when you are defining a new element then the `targetNamespace` is used. will be done in XML serialization of WSML.
26. It is bad form to put filenames in the namespaces, because this suggests that the namespace is not persistent  
[Rubén Lara] Not sure. Comments?  
[HL] Jos, I have seen both (checking daml library), but agree we should recommend Jos approach, see point 28.
27. I propose to do the following for the namespaces of the descriptions in this document: make the namespace independent of the deliverable number and deliverable version and \*let the mediators handle the versioning problem\*  
[HL] OK, the version does not have to be in the URI, since we have a (optional!) version tag
28. you can always import specific versions of ontologies, which use the version-independent `targetNamespace`  
[HL] Actually it can work if we use Namespaces without filename: [www.wsmo.org/ontologies/dateAndTime](http://www.wsmo.org/ontologies/dateAndTime) and at that location provide an overview page linking to different versions and formats (e.g. the wsml as text/plain and text/html).  
[MS] IMHO, this issue belongs to the concept / architecture for OO Mediators, not here.
29. You should not use the namespace prefix 'default'. Just put the namespace there without any prefix, this would be compliant with namespace specifications in XML  
[Rubén Lara] The things about sound good to me. Comments?  
[HL] OK.
30. You associate the wsml prefix with the d16.1 document. However, this document is just a grammar and does not contain specifications of the elements. It would be more appropriate to use the WSMO-standard document (d2)  
[Rubén Lara] I think he is right.  
[HL] OK, maybe it should be: <http://www.wsmo.org/2004/d2/>, but I discussed with Jos that it would be probably better to fully exclude this, since the trick with making key words bold and prefixing it with wsml is out of the scope of this particular syntax anyway...  
[MS] solved.
31. You often use the term 'set', which seems a keyword, but it's not written in boldface  
[Rubén Lara] Will be corrected
32. Please comment on the language you use for logical expressions. this is currently not clear to me. You seem to be able to plug in any language, because you just specify a literal?  
[Rubén Lara] I think that in principle we would be able to map to any language that has the same expressivity we use (which is the one of F-Logic). Am I

right?

[HL] Not sure, it was just because we were not able to parse it in the according version of D16.1. We should not use quotes I think, but stick to ONE particular syntax and semantics.

[MS] the WSMML representation of logical expressions is isomorph to F-Logic, thus the mappings to other languages starts from F-Logic (not from WSMML)

[HL] It is not completely isomorph! Explanations about the mapping (does and don'ts) have now their own section.

33. Long listings should not occur in the document itself, but rather in an appendix and the sections themselves should give some general descriptions and small examples from the listings, but prehaps this comment is more for later versions of the deliverable.

[Rubén Lara] Later versions.

34. Listing 2: there is a concept with no attributes, called 'instant'. What use is it to have a concept with no attributes?

[Rubén Lara] You have a detailed conceptual model by identifying this as separate concept. Although it has no properties, you can still have instances (Dependent on the answer from Dieter to my mail on the modeling of ontologies)

[HL] An instant is the superclass of all classes that represent a particular point in time.

35. Then I have something to say about variable definitions:

- WHY????

- It is not clear how the variables defined globally are used. The names of these variables seem to correspond with some parameters of functions, but the relation is not clear. Also, all parameters of the functions are defined and typed locally.

- the global variable declaration do not seem to have any purpose; they just confuse the reader

- There does not appear to be one global variables that is used. I only see variables with local scope in the axiom definitions.

[Rubén Lara] Already discussed.

[HL] Remark: Decision taken by Dieter to keep it.

36. Listing 2: I don't like this trick to define integrity constraints via a special 'invalid' predicate. This is an implementation issue, which should be handled when compiling the human readable syntax into FLORA-2 representation.

[Rubén Lara] Already discussed. Holgi, can you add an explanation?

[HL] Right, but Jos is right they should be removed and only have empty heads since it does not make sense to say this is invalid if one of its components is invalid, since the ontology has then no model anyway...

37. The use of the functions is not really clear to me. Furthermore, they seem to be implemented as predicates (?). It would also be useful to have some idea of the implementation of a function where you see the definition, because it is not at all clear which axiom is used to define which function.

[Rubén Lara] Already discussed

[HL] Same as point 10

38. Listing 2, function contains: X should be a parameter and the range of the function should be boolean, because you want to know if an instant is contained in an interval.

[HL] This was a mistake, clearly it should be a relation and not a function.

39. Listing 4: I'm not sure how many people can agree with the fact that Boesby is a city. Are you sure this ontology is shared?  
[HL]Admitted and changed - but it is beautiful anyway :)

---



\$Date

[webmaster](#)